AN12709

恩智浦触摸软件开发指南

版本 0 — 2019 年 1 月 20 日

应用笔记

目录

1 恩智浦触摸

恩智浦触摸软件能够加快您的触摸应用程序的开发速度，非常适合与 Kinetis® MCU 配套使用。该软件下载提供源代码，具有触摸检测算法，非常适合基于 RTOS 的应用程序。恩智浦触摸软件采用模块化架构，其中包含各种以触摸为中心的控件，模块和电极数据对象，支持集成和自定义的功能。

恩智浦触摸软件的目标是帮助客户在触摸应用程序中实现顺利迁移和简单设计。

这是针对触摸控制应用的整体解决方案，恩智浦触摸提供触摸软件和硬件触摸感应 IP TSI。

软件库与 Kinetis KE1x 平台的结合带来了很多改进，如下所示:

- 出色的 EMC 能力、抗噪能力，均通过 IEC61000-4-6 标准测试和 3 V/10 V 测试。
- 支持自电容和互电容模式，最多支持 6 × 6 矩阵触摸板。
- 耐液体、水、油、冷蒸汽等方面性能优异。
- 可配置的灵敏度高，支持高达 10 mm 厚的玻璃/塑料覆盖层。

2 触摸感应硬件支持

基于电荷转移物理原理，恩智浦为 Kinetis KE1x 设备提供最新的 TSI v5 硬件外设。这种方法在环境变化和 EMC 的敏感性和抗扰性方面提供了合适的性能。触摸感应演示软件示例针对带有插入的 FRDM-TOUCH 模块的 FRDM-KE15z，不包括在 FRDM 板中，须单独订购。

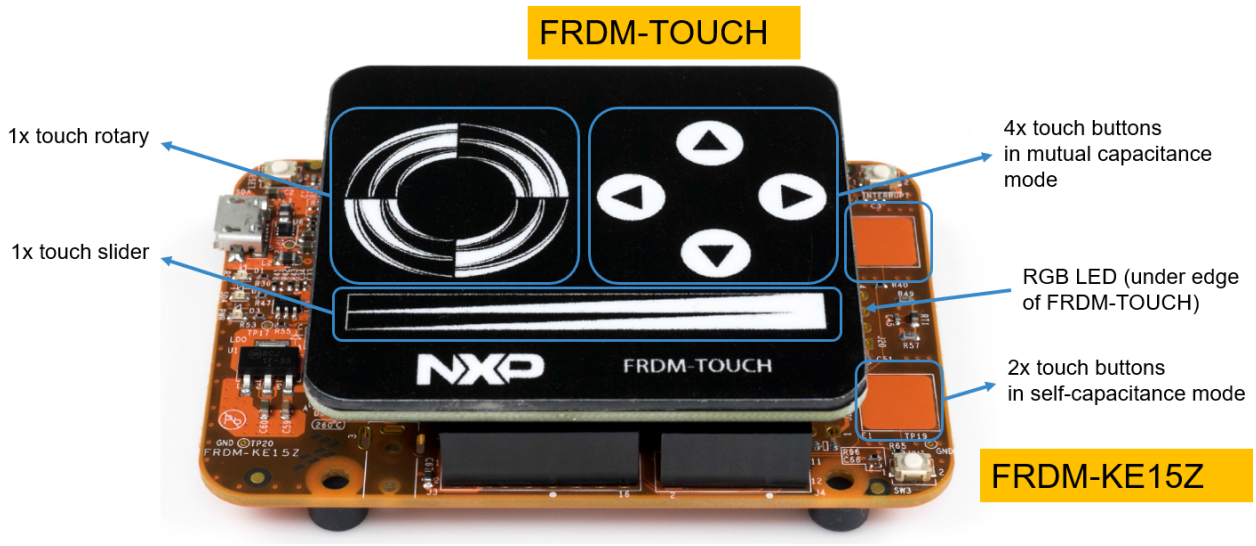


图 1. FRDM-KE15Z 和 FRDM-TOUCH

3 NT 软件库

恩智浦提供了一个免费的软件库，它有触摸应用开发需要的所有软件，不仅可以检测触摸，还可以实现更高级的控制套件，如滑块或小键盘。

TSI 背景算法可用于触摸键盘和模拟解码器，灵敏度自动校准，低功耗，高精度，耐水性。软件以“目标 C 语言代码结构”的源代码形式发布。

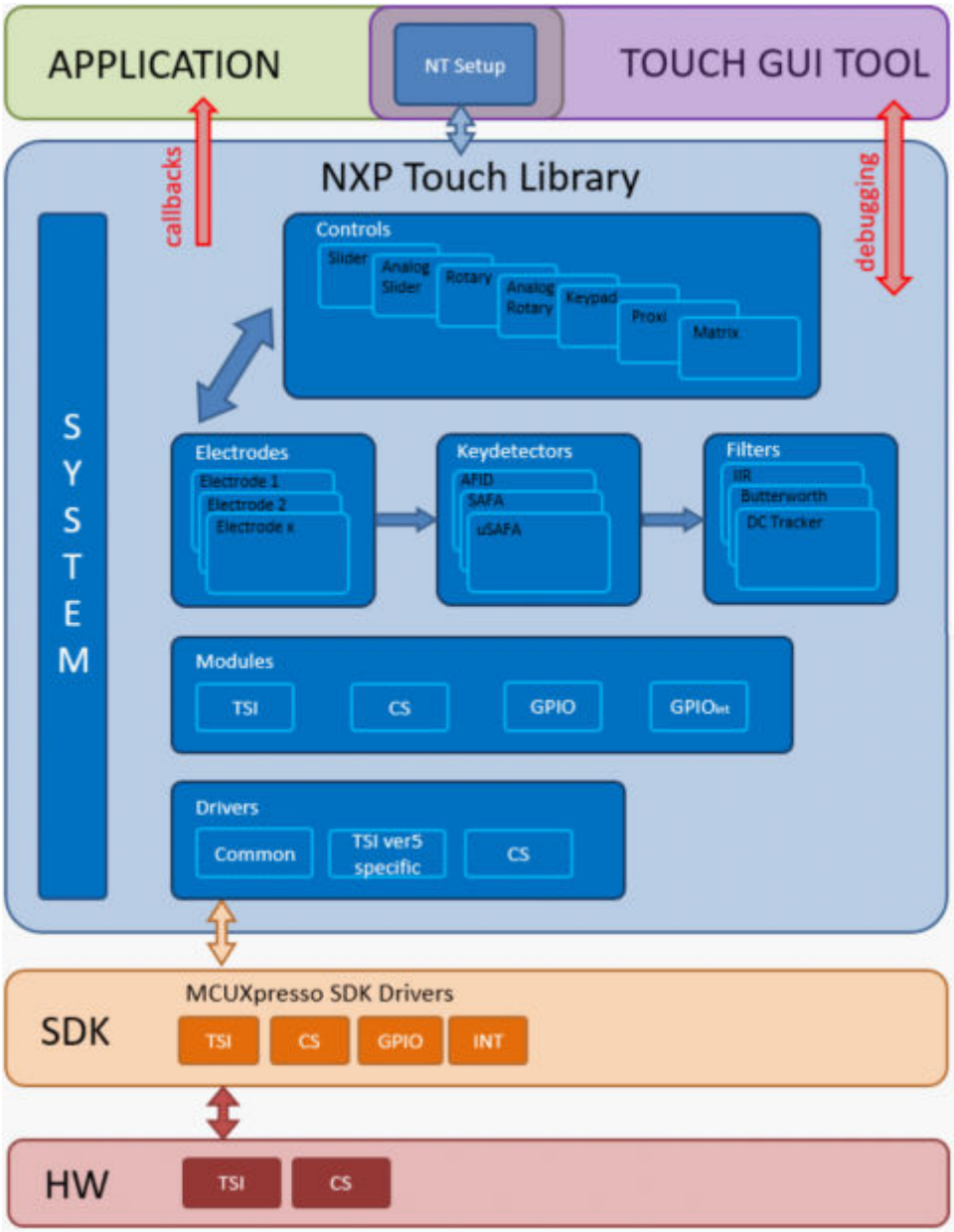


图 2. 恩智浦触摸软件库框图

4 FreeMASTER 运行时调试工具

FreeMASTER 软件是恩智浦提供的可免费下载的调试和可视化工具。

通过调试端口（Kinetis 的 JTAG/SWD）或串口（UART）接口支持与目标处理器的通信。使用 JTAG/SWD 连接更快，不需要额外的 UART 连接。

目标 MCU 的串行端口驱动代码可用于大多数 MCU 目标。项目中的任何变量都可以轻松监控和修改（基于“目标方寻址 TSA”表读取内存）。

可创建基于 HTML/JavaScript 的 GUI 以获得更多功能。

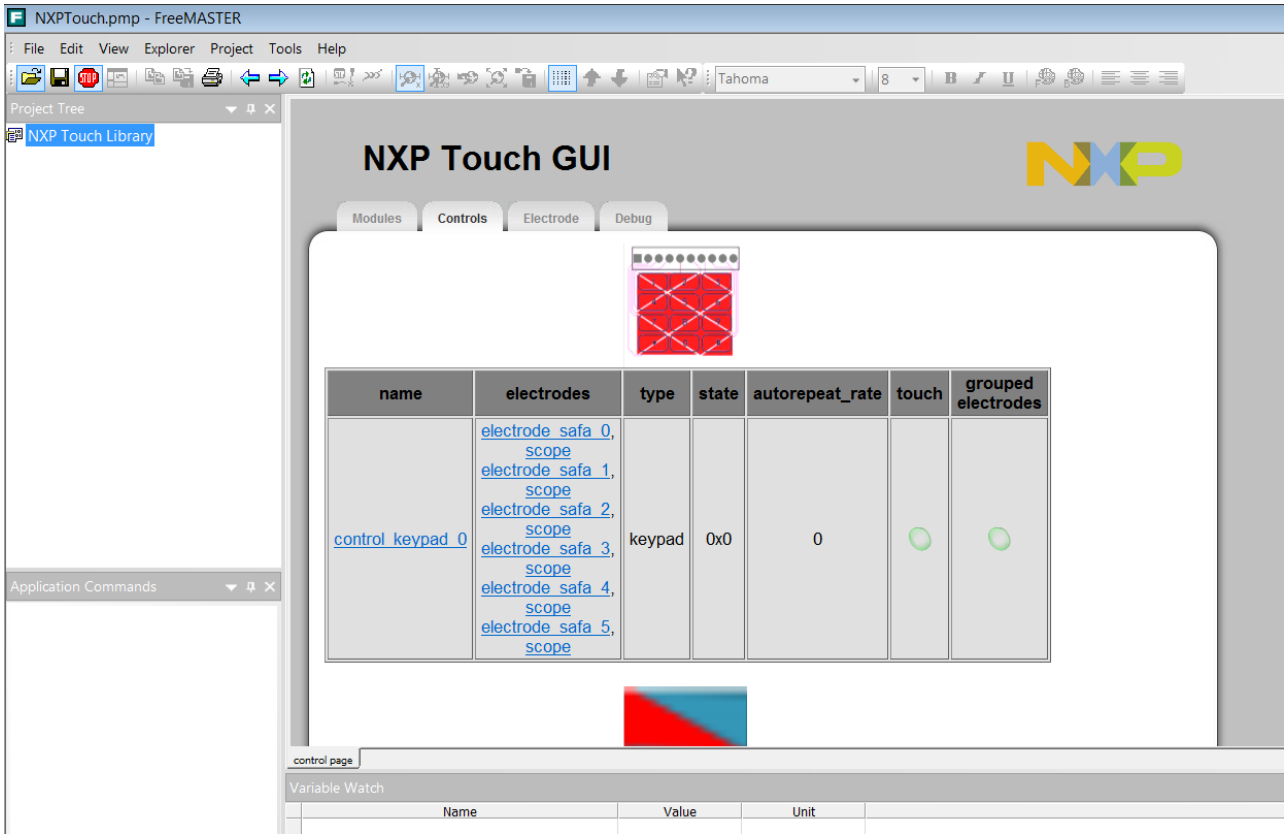


图 3. FreeMASTER 中的 NXP Touch GUI

5 支持的编译器

恩智浦触摸软件库作为一个可选的中间件组件，可在 Kinetis SDK 下载期间选择。在生成 SDK 包之前，必须选择合适的编译器。

支持的编译器 IDE 如下：

- IAR EWARM
- Keil uVision
- 恩智浦提供免费的 MCUExpresso

5.1 下载并安装 MCUXpresso

MCUExpresso 可以从 MCUXpresso Software and Tools 下载。

请按照以下步骤安装 IDE。

- 如果尚未注册，请创建登录名。
- 下载安装程序。
- 安装软件。
- 启动 MCUXpresso。

6 软件下载

请按照以下步骤来构建和下载 FRDM-KE15z 的 SDK：

1. 进入 MCUXpresso SDK Builder 网页。
2. 登录到恩智浦。
3. 选择开发板。
4. 从 Boards/Kinetis 菜单中选择 FRDM-KE15Z。
5. 构建 SDK。

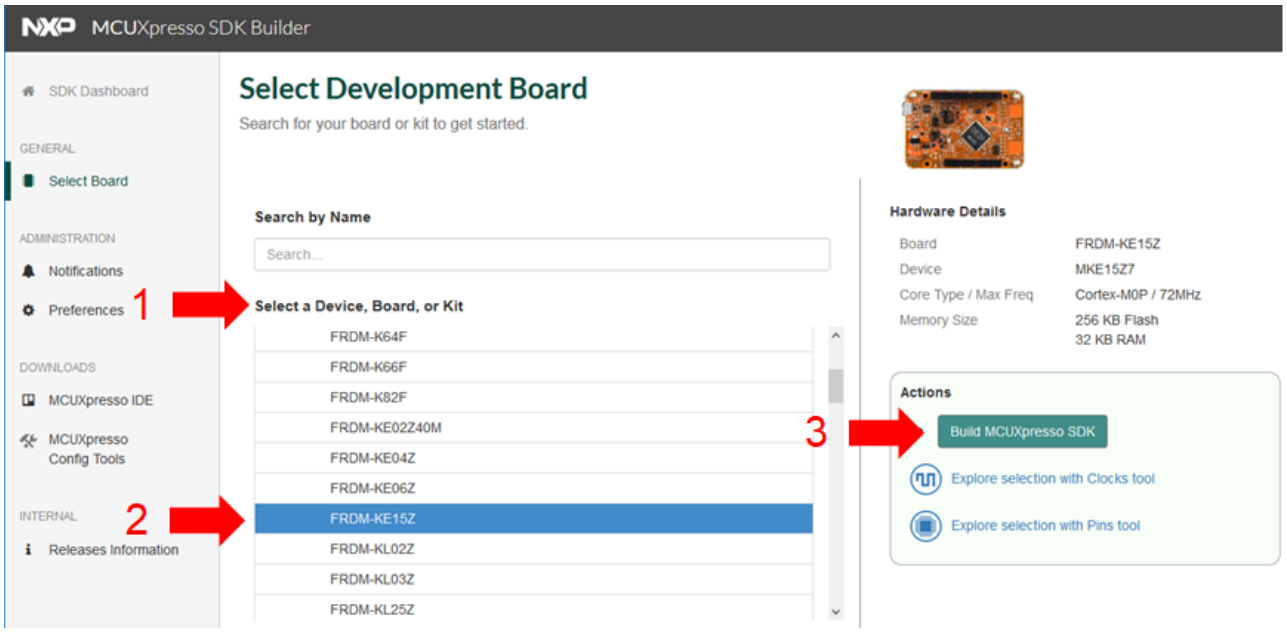


图 4. 建立 SDK 套件

6.1 添加触摸支持到 SDK

请按照以下步骤添加“ Touch”可选中间件（软件组件），并选择适当的工具链/ IDE。您可以为单个工具链/ IDE 构建软件包，也可以选择支持的所有工具链。

1. 添加软件组件（可选的中间件）。
2. 选择触摸。
3. 保存更改。
4. 注意 SDK 版本。
5. 在前缀中使用版本名作为存档名称。
6. 请求建立。

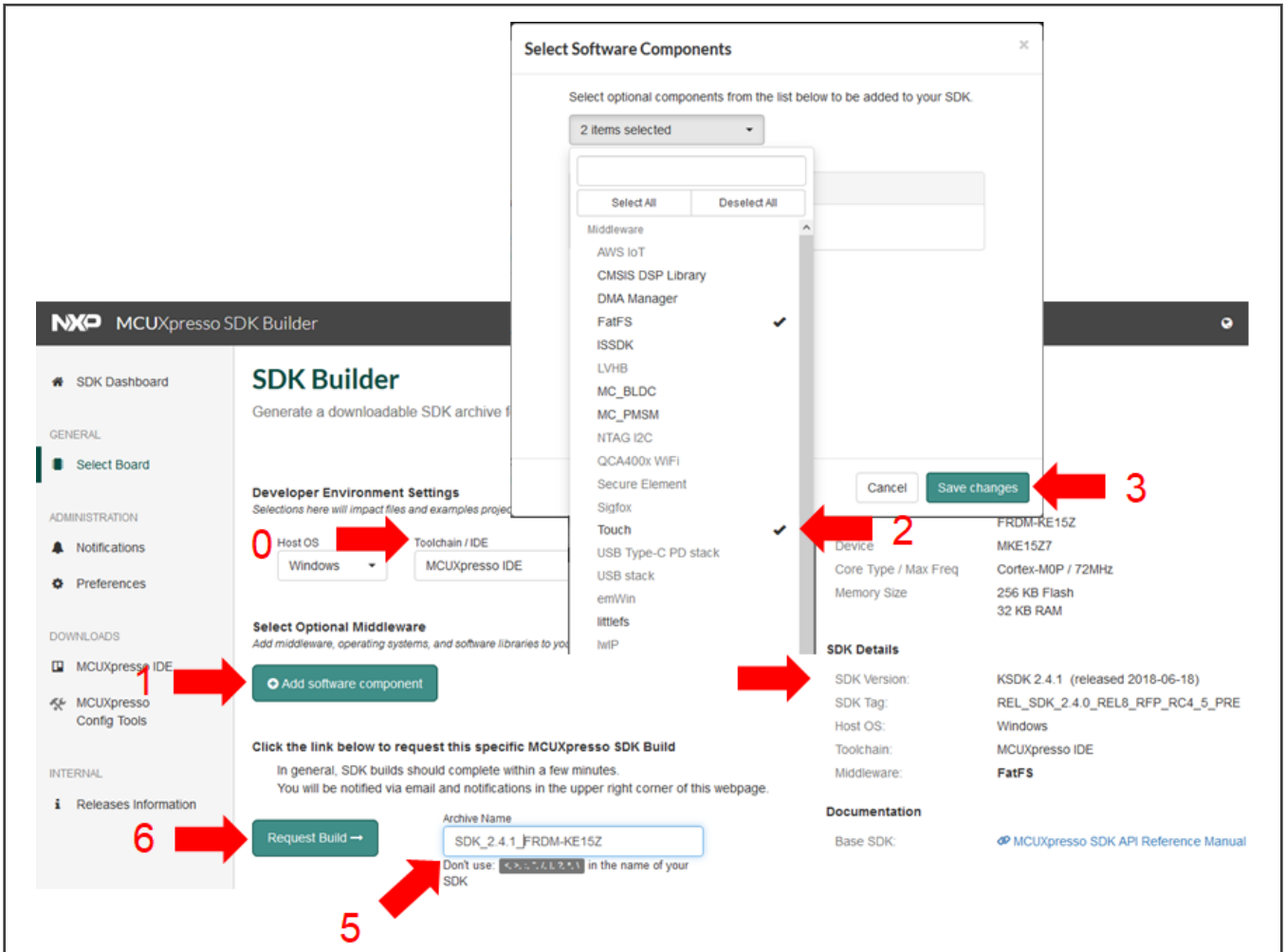


图 5. 向 SDK 添加 Touch 支持

6.2 下载 SDK 和文档

1. 下载 SDK 存档。
2. 下载 SDK 文档。
3. 同意 EULA。

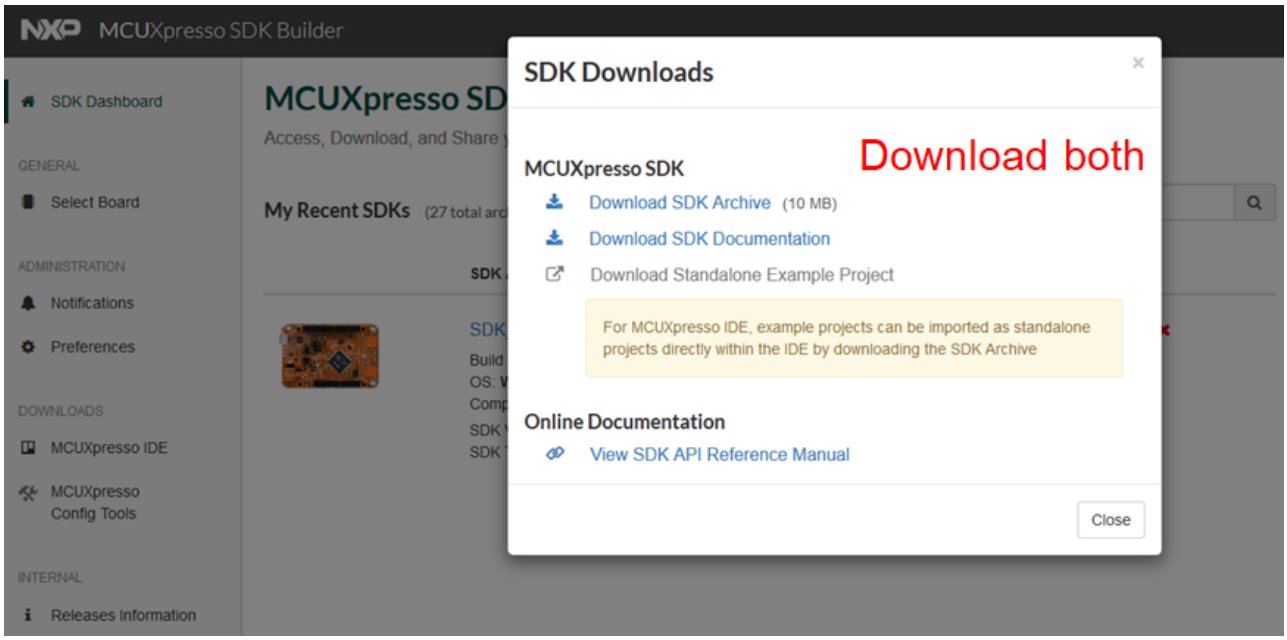


图 6. SDK 下载

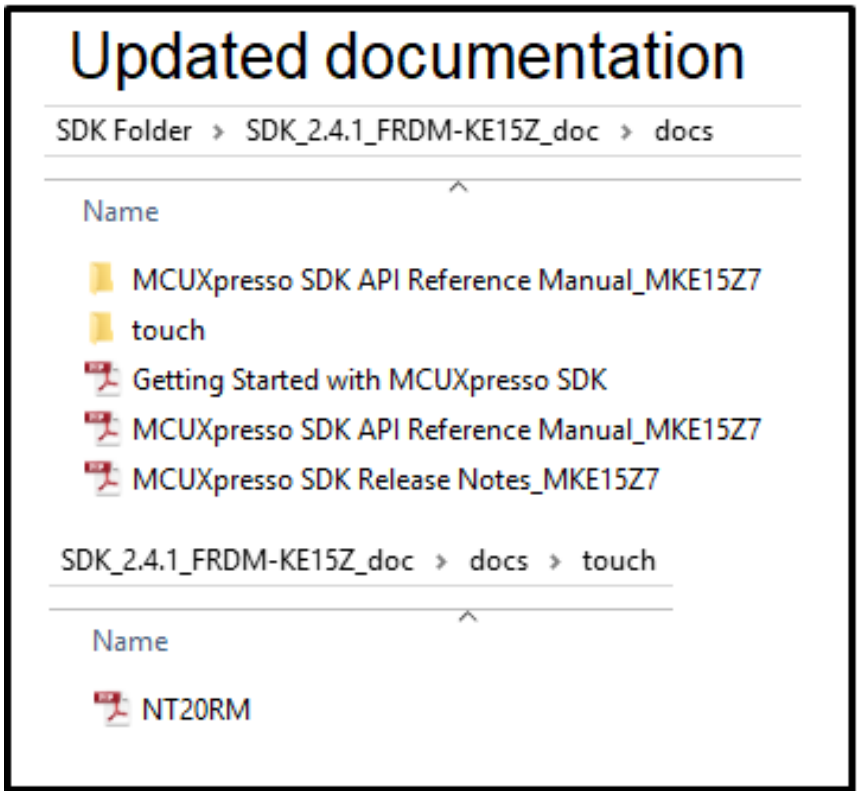


图 7. SDK 文档下载

6.3 FreeMASTER 下载和安装

可以从 FreeMASTER Run-Time Debugging Tool 免费下载最新版本的 FreeMASTER。

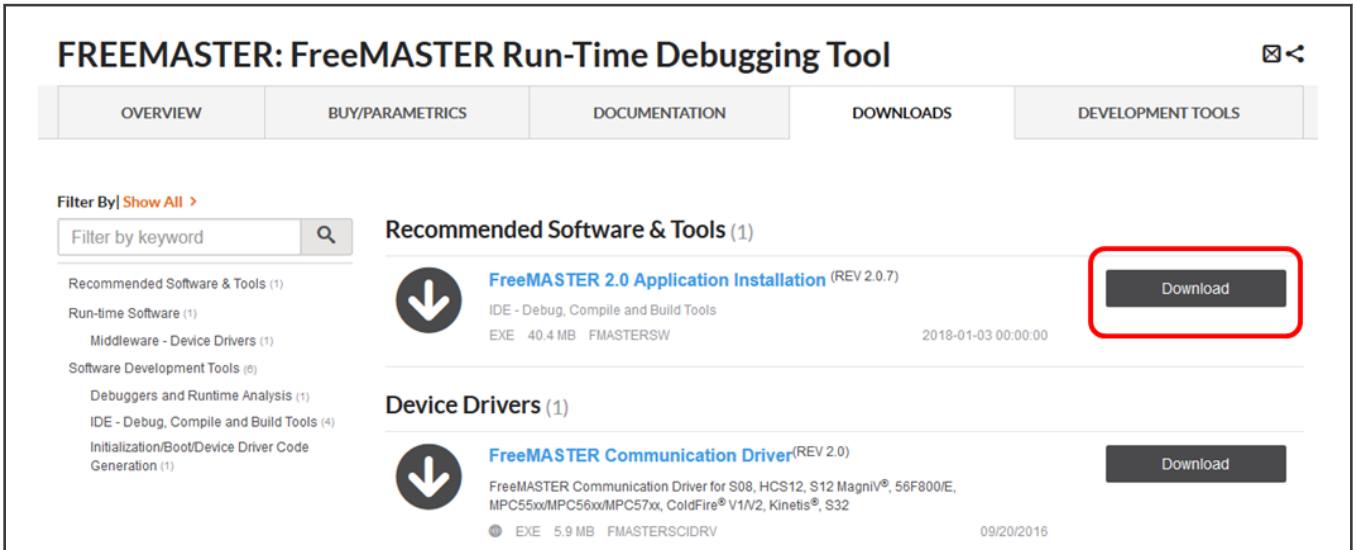


图 8. FreeMASTER 应用程序下载

7 从 FRDM 板和触摸演示开始

7.1 FRDM 板设置

- 确保 J15 跨位置 2 和 3 使能 3.3 V 供电。
- 将 FRDM-TOUCH 板连接到 FRDM-KE15Z。把 USB 电缆连接到你的电脑上，给 Freedom 板上电。
- 下载“touch_sensing”到开发板上。

7.2 触摸感应演示示例

1. 如软件下载所述，从 MCUXpresso 网站下载最新的 FRDM-KE15Z SDK 软件包。
2. 启动 MCUXpresso 。
3. 将 SDK 包（.zip）拖到 MCUXpresso 中的 **Installed SDKs** 视图中以安装 SDK。

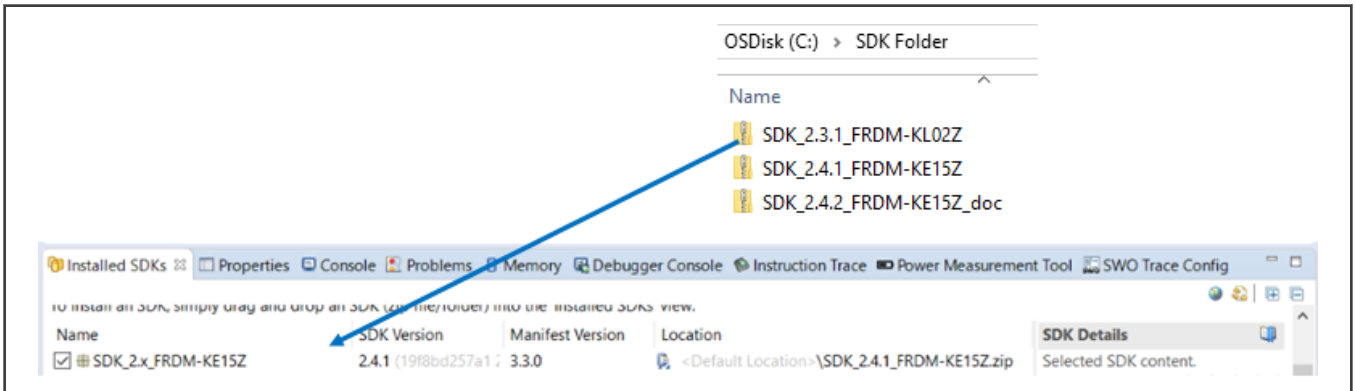


图 9. 将 KE15Z SDK 导入到 MCUXpresso

7.2.1 导入 touch_sensing 演示

请按照以下步骤将软件示例项目导入到 MCUXpresso IDE。

1. 导入 SDK 示例（touch_sensing）。

2. 选择 **frdmke15z**，单击 **Next**。
3. 展开 **demo_apps**。
4. 检查 touch_sensing，然后单击 **Next**。

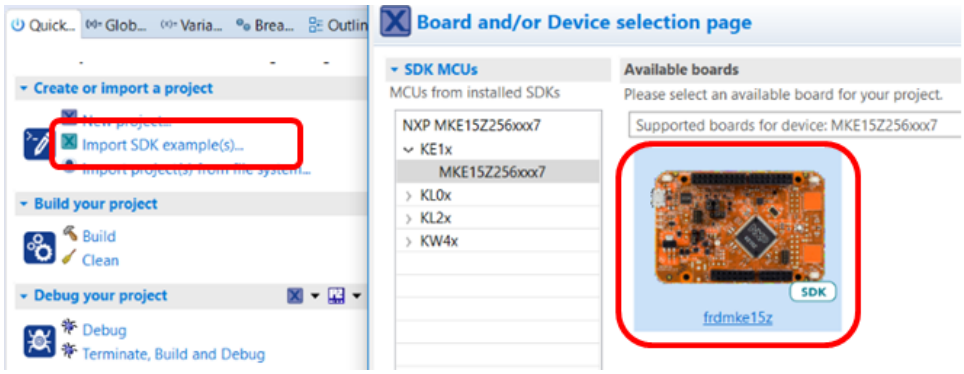


图 10. 导入 SDK 示例

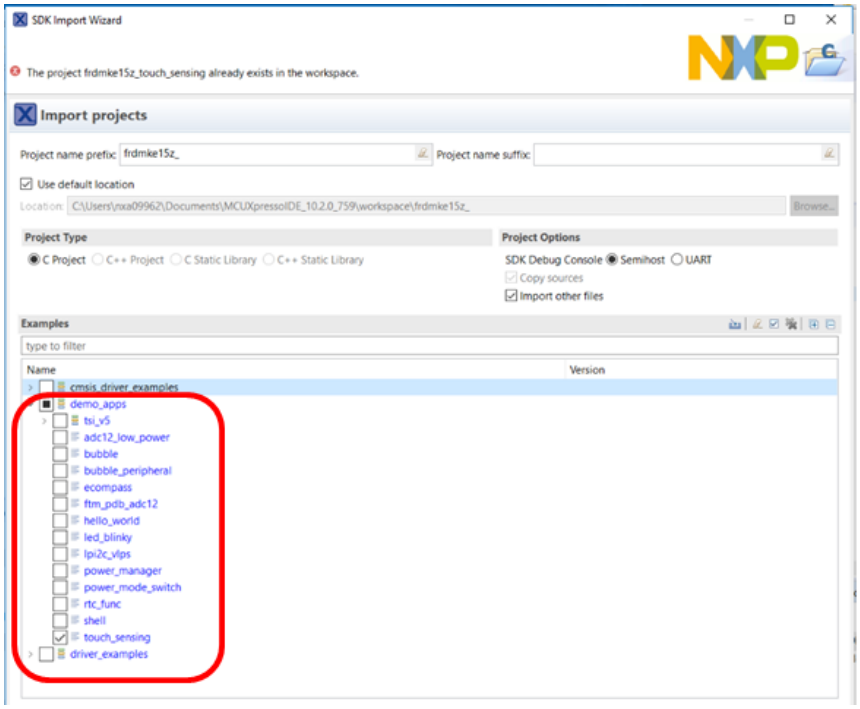


图 11. 导入触摸感应演示

7.2.2 运行和调试触摸感应示例

1. 在 **MCUXpresso IDE Project** 窗口中，展开 **frdmke15z_touch_sensing**。
2. 双击 main.c。
3. 编译和调试触摸感应项目（以将应用程序加载到 Flash 中）。
4. 单击 **Resume** 图标以运行演示。

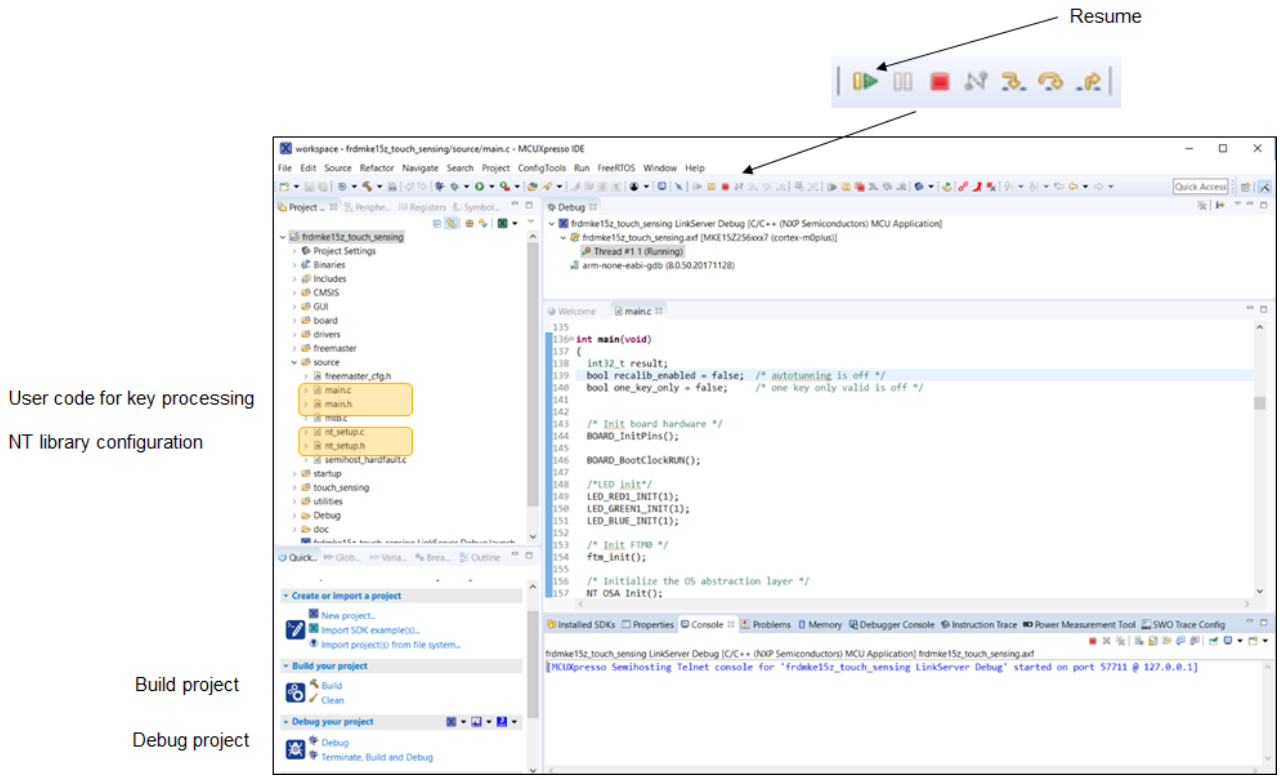


图 12. 调试 demo

7.2.3 探索 touch_sensing 演示

- 按 FRDM-KE15Z 上的 E1 键 — RGB LED 变为黄色。
- 按 FRDM-KE15Z 上的 E2 键 — RGB LED 变为青色。
- 按 FRDM-TOUCH 的 Up 键— RGB LED 变成黄色。
- 按 FRDM-TOUCH 的 Right 键 — RGB LED 变成绿色。
- 按 FRDM-TOUCH 的 Down 键 — RGB LED 变成蓝色。
- 按 FRDM-TOUCH 的 Left 键 — RGB LED 变成白色。
- 从右向左滚动 FRDM-TOUCH 上的滑块 — 降低 RGB 强度。
- 围绕 FRDM-TOUCH 旋转滚动 — RGB 色调改变。

7.3 独立的 FreeMASTER GUI

在 FRDM-KE15Z 上安装 FRDM-TOUCH 并将 flash_sensing 演示加载到 Flash 中的情况下，在开发板和 PC 之间连接 USB 电缆。

- 所有的触摸电极都应该通过改变 RGB LED 上的颜色或强度来响应。
  - 启动 FreeMASTER。
  - 将 *NXP Touch KE15Z.pmp* 文件拖到 FreeMASTER 窗口上。
- **Project Tree** 更改为 **NXP Touch Library**。

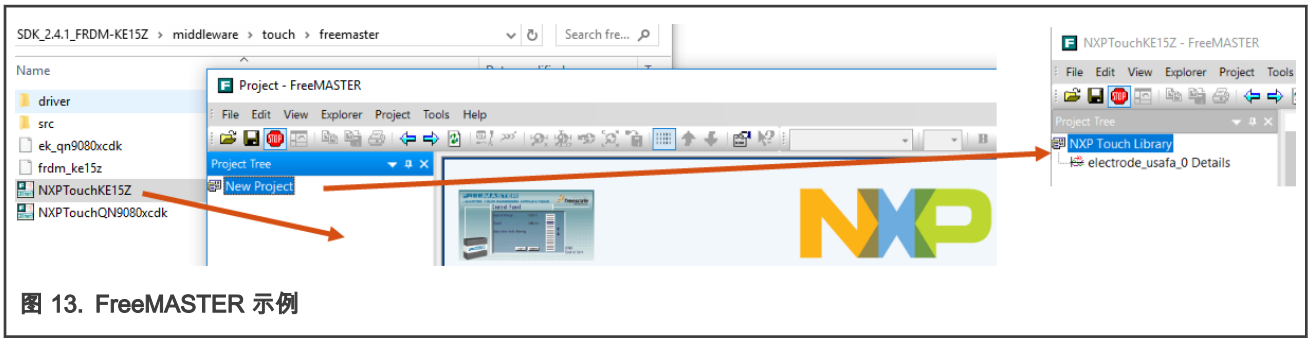


图 13. FreeMASTER 示例

7.3.1 FreeMASTER GUI 连接

Freemaster 支持多种通信协议，如串行 COMx 或调试接口上的直接连接。按照以下步骤，通过板载 mbed USB 连接到串口。

1. 从工具菜单中选择连接向导，然后单击 **Next**。
2. 使用直接连接到板载 USB 端口，单击 **Next**。
3. 确保选择 mbed 串行端口（COMxx），单击 **Next**。
4. 将检测到 **COMxx UART** 端口，选择 **Yes, Finish**。

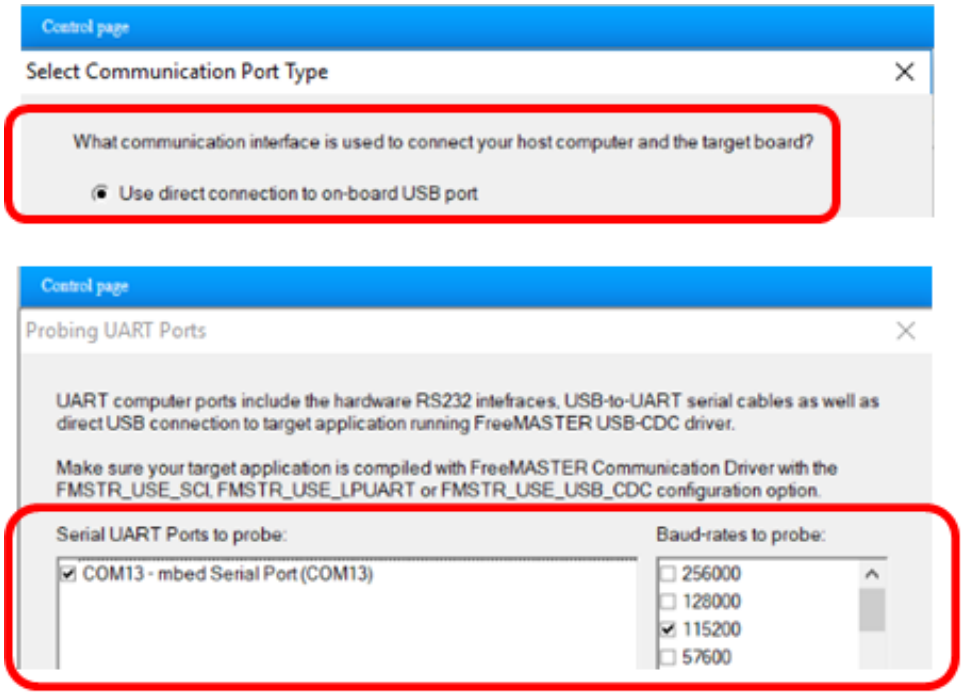


图 14. 设置 FreeMASTER 通讯

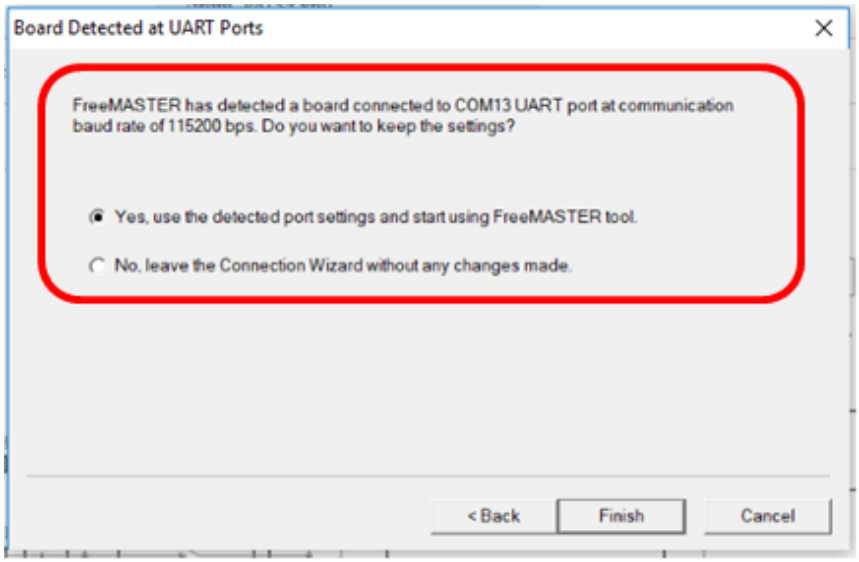


图 15. 选择 UART

7.3.2 TOUCH SW LAYERS 选项卡

要查看关于触摸软件组件的信息，在 **Control page** 窗口中选择 **TOUCH SW LAYERS** 选项卡，然后单击 **READ CONFIGURATION FROM BOARD**，等待上传完成。

您可以向下滚动以查看：

- NT CONTROLS 控制

  Keypad_1，Aslider_2，Arotary_3 和用于每个控件的电极。

- NT ELECTRODES 电极

  所有使用的 12 个电极的信息如 图 16。

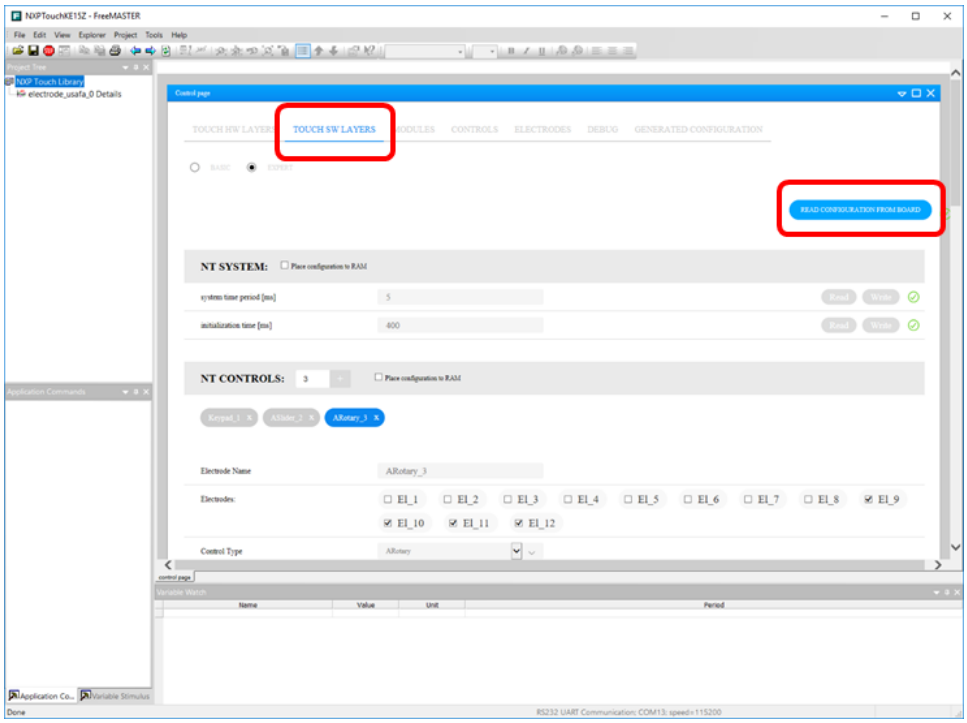


图 16. FreeMASTER - TOUCH SW LAYERS 选项卡

7.3.3 MODULES 选项卡

如果在控制页面窗口中选择 MODULES 选项卡，那么 sample project 中定义的所有 12 个电极都会显示出来。信号值在扫描过程中会改变。当 FRDM-TOUCH 上的向上箭头被按下时，电极 2 为有效触摸。请注意无论软件设置中定义的电极名称如何，电极均从零开始索引。

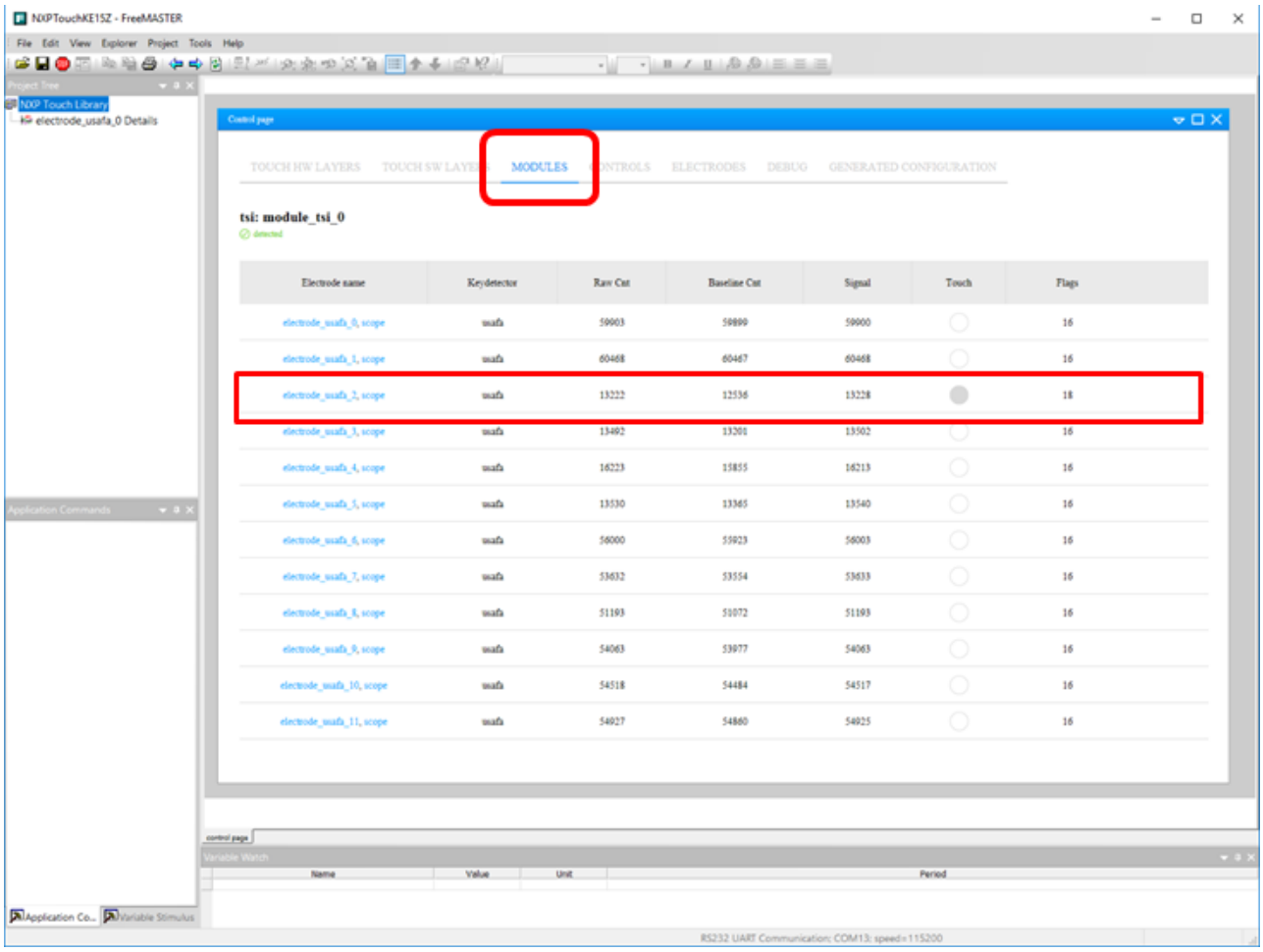


图 17. FreeMASTER - MODULES 选项卡，电极视图

```
const struct nt_electrode * const nt_tsi_module_electrodes[] = {
  &El_1,  &El_2,  &El_3,  &El_4,  &El_5,  &El_6,  &El_7,  &El_8,  &El_9,  &El_10,  &El_11,  &El_12, NULL
};
```

图 18. “nt_setup.c” 中的模块电极分配

7.3.4 CONTROLS 选项卡

如果选择了控制页面窗口中的 **CONTROLS** 选项卡，示例项目中的三个触摸控件（键盘、旋转和滑块）就会显示出来。

当 FRDM-TOUCH 上的任何一个键盘电极被按下时，键盘控制将显示一个有效的触摸。

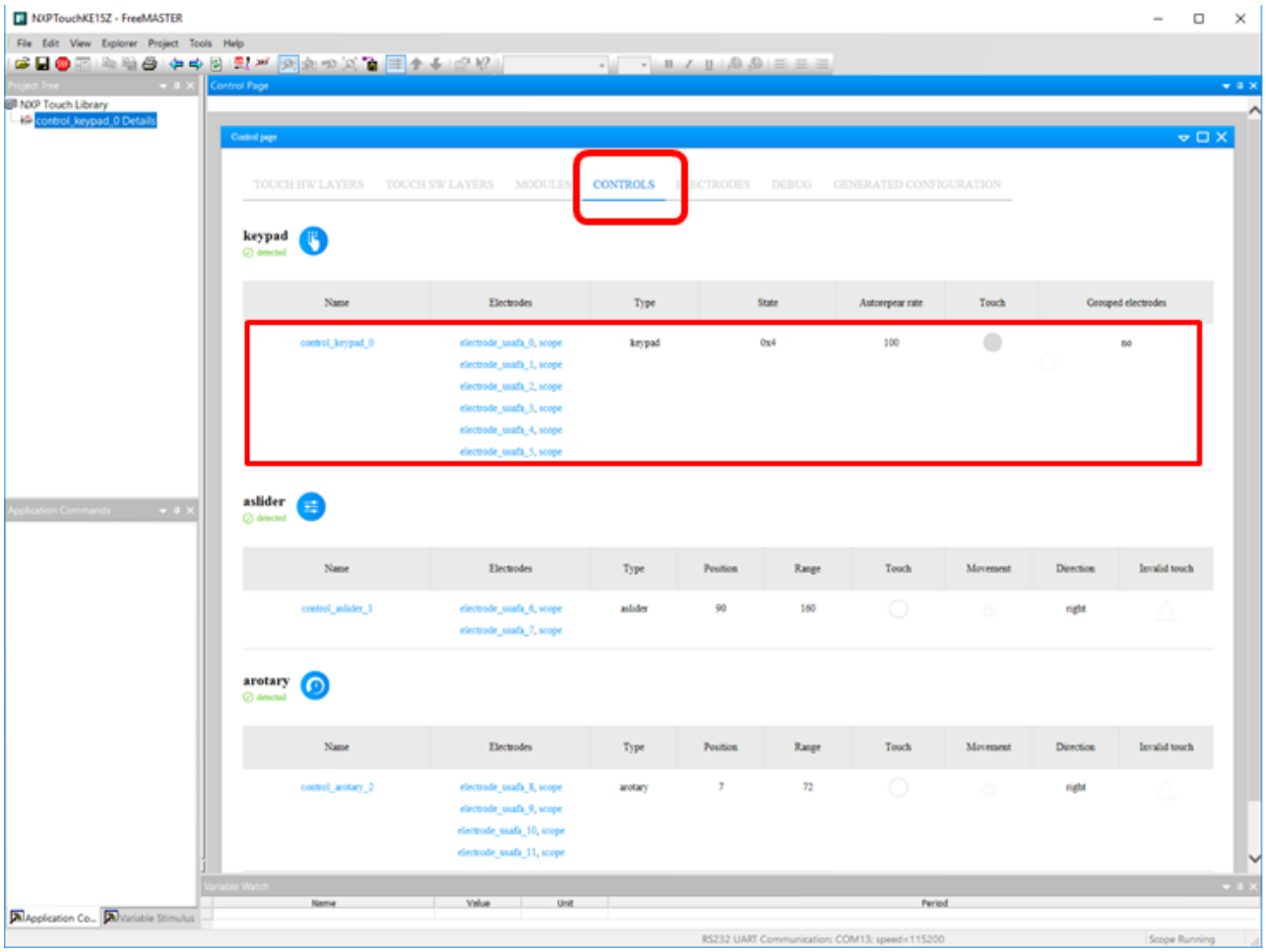


图 19. CONTROLS 选项卡 - keypad

7.3.5 ELECTRODES 选项卡

可以在 **ELECTRODES** 选项卡中监视有关电极触摸/释放事件的详细信息。请按照以下步骤操作：

1. 在 **CONTROLS** 页面窗口中选择 **ELECTRODES** 选项卡。
2. 在 **SELECTED ELECTRODE** 下拉菜单选择 **electrode_usafa_1**。
3. 按住 **FRDM-KE15Z** 上的 **E2** 电极，状态 0（或 2）显示有效触摸。
4. 释放 **E2** 电极，在状态 1（或 4）中显示有效释放。

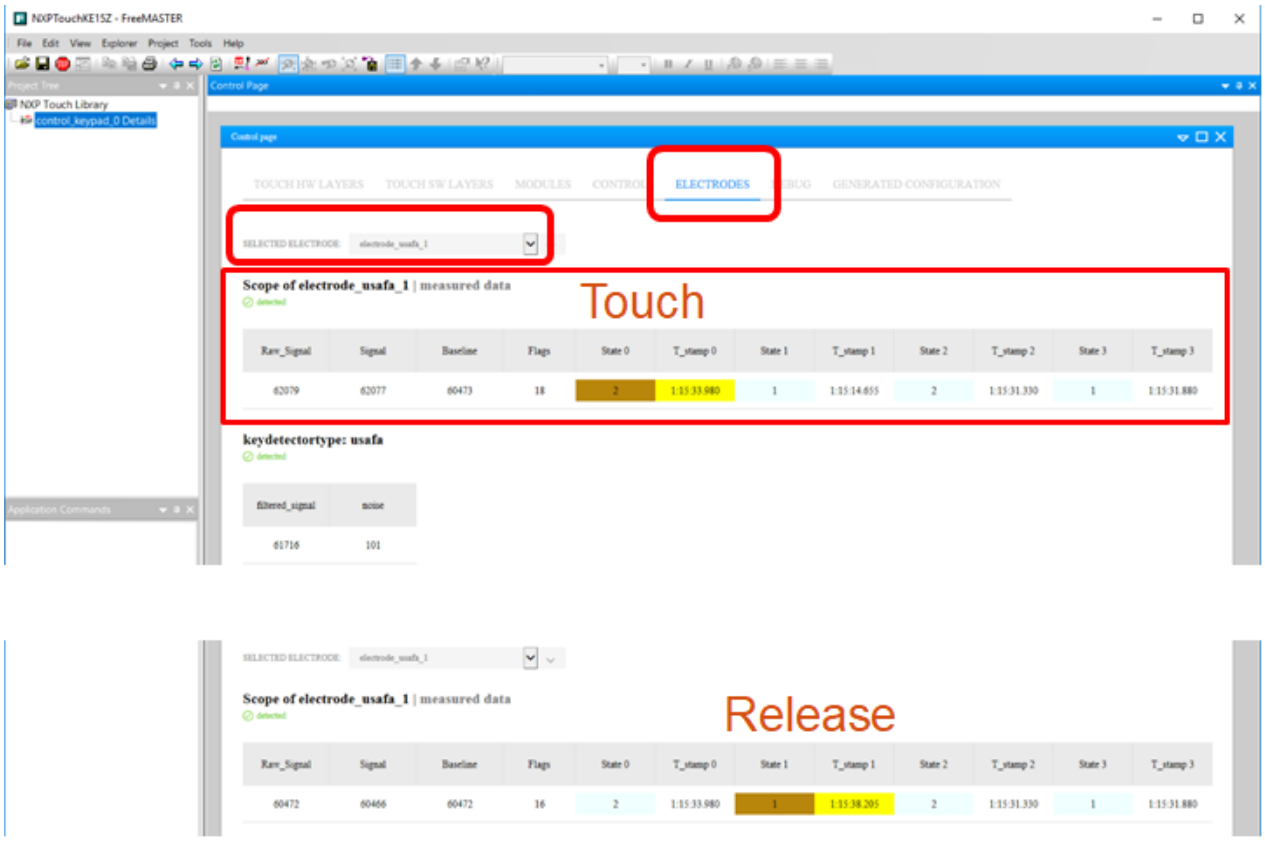


图 20. ELECTRODES 选项卡 — 触摸和释放事件

7.3.6 FreeMASTER 示波器视图

点击 **Scope** 后，可以监控大部分重要的触摸传感信号和按键检测值。但是只能选择和监视单个电极。请按照以下步骤观察所选电极的信号，有以下几种选择：

- 在 **MODULES** 选项卡中，单击电极的 scope 链接。

  （如果您错误地点击 **electrode_usafa_x**，就会出现错误，只需点击 **Yes**。）
- 在 **CONTROLS** 选项卡中，单击控件的名称以查看该控件中的所有电极。
- 在 **CONTROLS** 选项卡中，单击 scope 链接以获取单个电极。
- 在 **ELECTRODES** 选项卡中，从滚动窗口中选择一个电极，然后单击 **CONTROLS** 页面窗口底部的示波器。

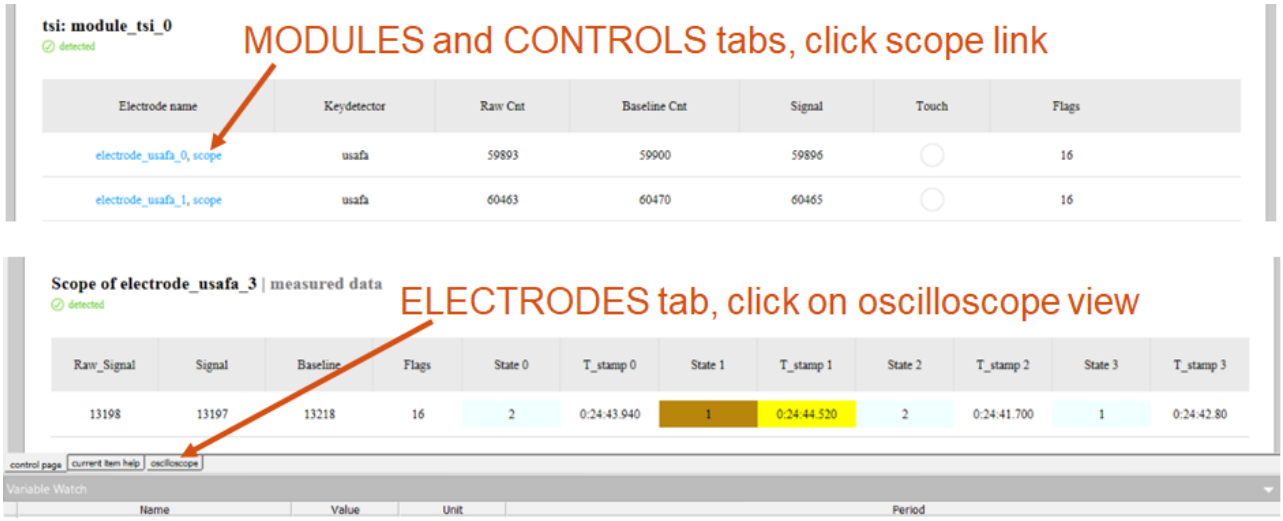


图 21. 进入示波器视图

7.3.7 单电极示波器视图

所有重要的关键探测器信号都可以在示波器窗口中看到，因此可以轻松调整单个电极的灵敏度和触摸阈值。当按压 FRDM-TOUCH 上的电极 2（上箭头按键）时，可以看到信号步长和触摸检测。

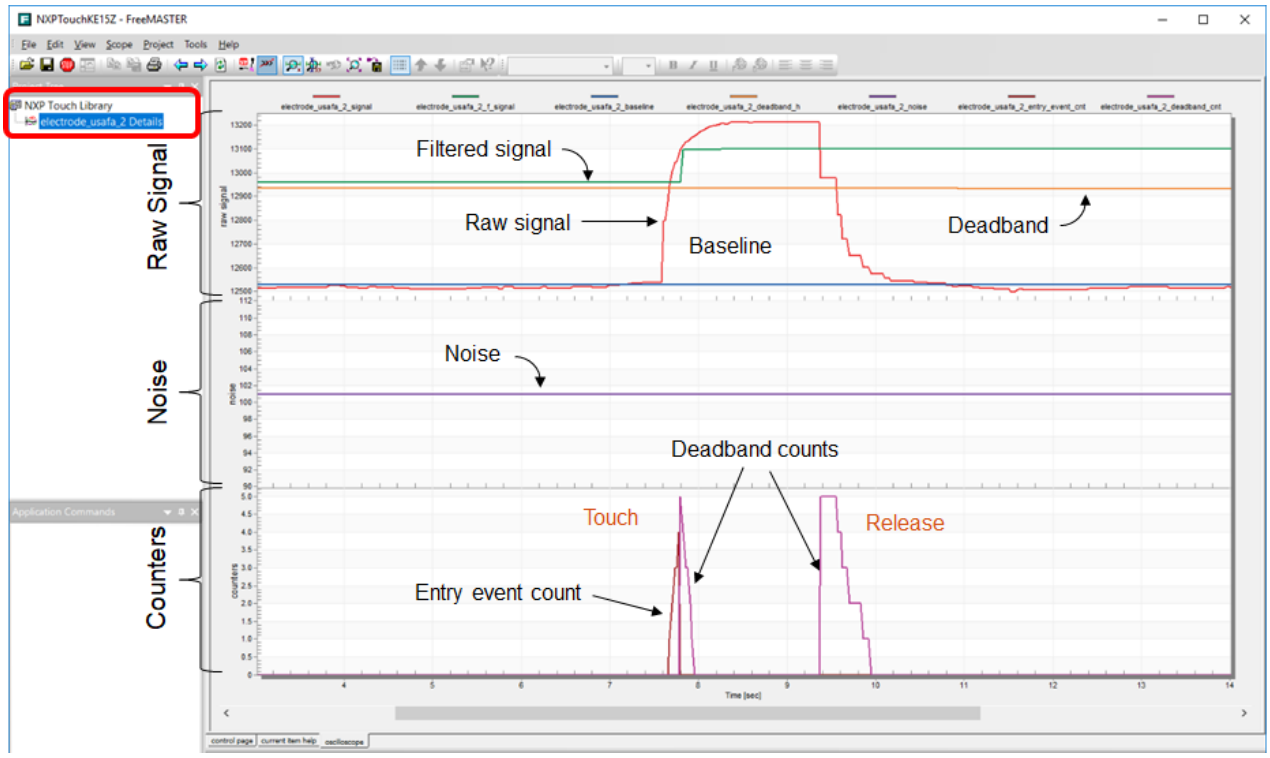


图 22. 单电极示波器视图 — 按键检测器信号

7.3.8 滑块控制范围视图

滑块检测到的位置，键盘按钮状态标志这样的值也可以被监视。

例如，模拟滑块由两个电极组成。如果我们从左到右滑动手指，会看到 0 号电极和 1 号电极上的信号变化。位置和方向由两个电极信号计算得出。

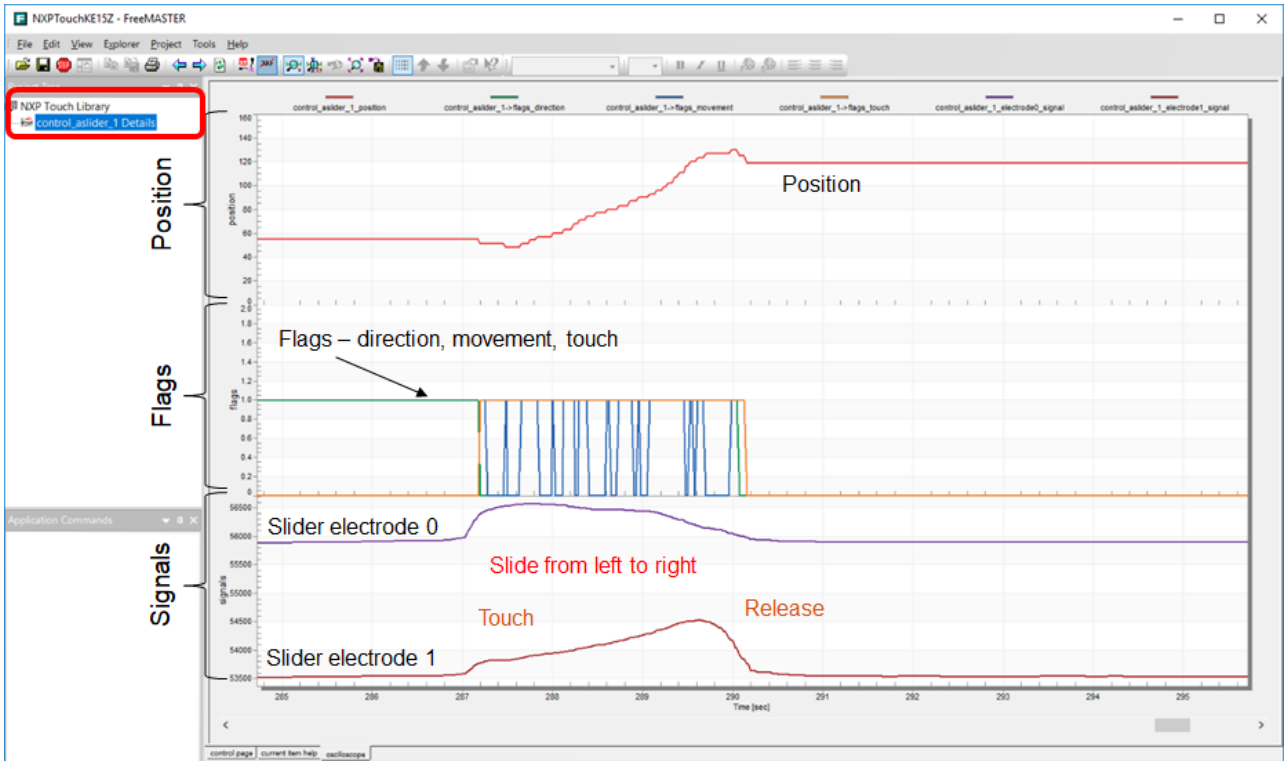


图 23. 单电极示波器视图 — 按键检测器信号

7.4 触摸传感 demo 软件配置

7.4.1 FRDM-TOUCH

FRDM-KE15Z 上的 RGB LED 响应以下触摸：

- 四个按键
- 触摸转盘
- 触摸滑块

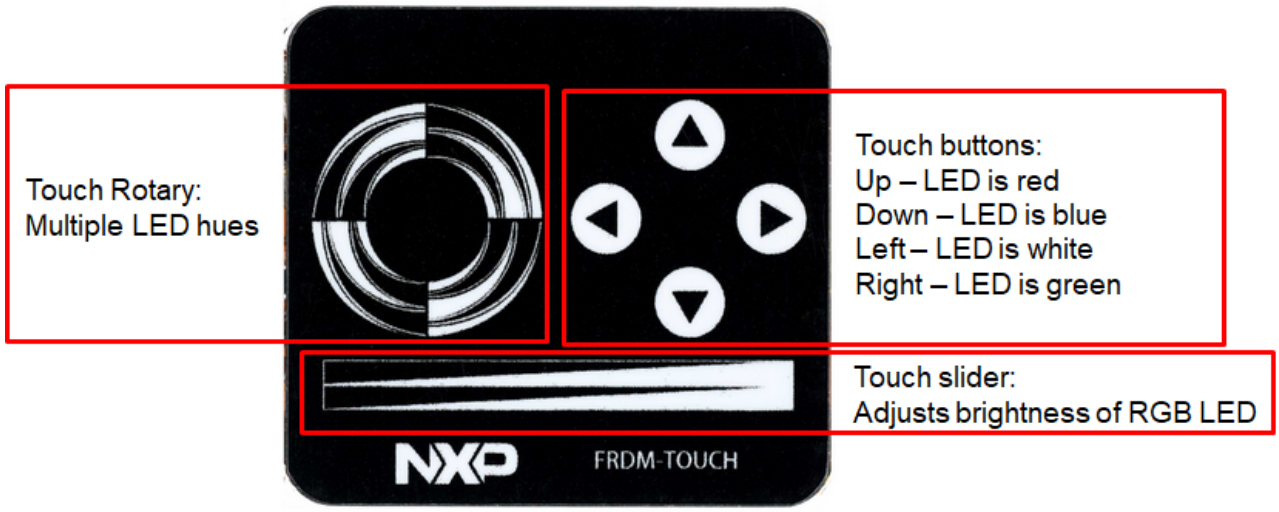


图 24. FRDM-TOUCH 控件

7.4.2 TSI 通道分配

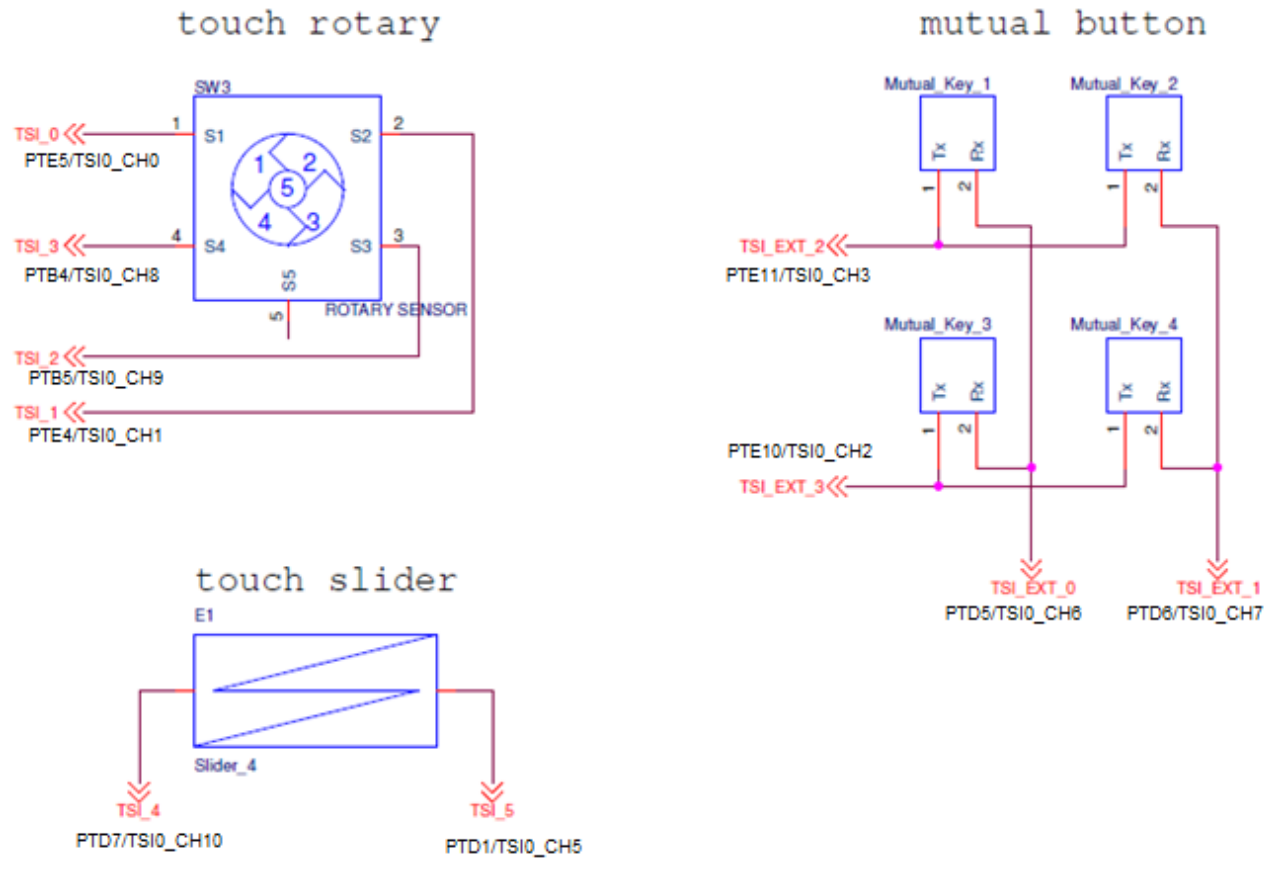


图 25. TSI 输入原理图

7.4.2.1 在软件中分配 TSI 通道

触摸演示软件基于 Kinetis SDK。使用 SDK 的好处是 SDK 支持 MCU 的所有外围器件的驱动，例如 UART，Timers，SPI 驱动等。

但是，触摸库仅使用了 SDK 中的少量文件。主要用到 TSI 和计时器的底层驱动程序。但是这些文件可以由用户定义的驱动程序代替。这意味着可以在 Kinetis SDK 上独立地将库、源文件轻松集成到另一个项目。

在 Kinetis SDK 中，物理 TSL 通道的虚拟电极分配在 “board.h” 文档中完成。

```
/* Push buttons - mutual electrodes */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_ELECTRODE_1  NT_TSI_TRANSFORM_MUTUAL(FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_1,FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_1)//TF_TSI_MUTUAL_CAP_TX_CHANNEL_3
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_ELECTRODE_2  NT_TSI_TRANSFORM_MUTUAL(FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_2,FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_2)//TF_TSI_MUTUAL_CAP_TX_CHANNEL_3
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_ELECTRODE_3  NT_TSI_TRANSFORM_MUTUAL(FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_3,FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_3)//TF_TSI_MUTUAL_CAP_TX_CHANNEL_2
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_ELECTRODE_4  NT_TSI_TRANSFORM_MUTUAL(FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_4,FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_4)//TF_TSI_MUTUAL_CAP_TX_CHANNEL_2

#define FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_1  TF_TSI_MUTUAL_CAP_TX_CHANNEL_3      /* PTE11 */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_1  TF_TSI_MUTUAL_CAP_RX_CHANNEL_6      /* PTD5  */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_2  TF_TSI_MUTUAL_CAP_TX_CHANNEL_3      /* PTE11 */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_2  TF_TSI_MUTUAL_CAP_RX_CHANNEL_7      /* PTD6  */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_3  TF_TSI_MUTUAL_CAP_TX_CHANNEL_2      /* PTE10 */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_3  TF_TSI_MUTUAL_CAP_RX_CHANNEL_6      /* PTD5  */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_TX_ELECTRODE_4  TF_TSI_MUTUAL_CAP_TX_CHANNEL_2      /* PTE10 */
#define FRDM_TOUCH_BOARD_TSI_MUTUAL_RX_ELECTRODE_4  TF_TSI_MUTUAL_CAP_RX_CHANNEL_7      /* PTD6  */

/* Slider - self electrodes */
#define FRDM_TOUCH_BOARD_TSI_SLIDER_ELECTRODE_1  TF_TSI_SELF_CAP_CHANNEL_10     /* PTD1 */
#define FRDM_TOUCH_BOARD_TSI_SLIDER_ELECTRODE_2  TF_TSI_SELF_CAP_CHANNEL_5      /* PTD7 */
#define FRDM_TOUCH_BOARD_TSI_1  TF_TSI_SELF_CAP_CHANNEL_22
#define FRDM_TOUCH_BOARD_TSI_2  TF_TSI_SELF_CAP_CHANNEL_23

/* Rotary - self electrodes */
#define FRDM_TOUCH_BOARD_TSI_ROTARY_ELECTRODE_1  TF_TSI_SELF_CAP_CHANNEL_0      /* PTE5 */
#define FRDM_TOUCH_BOARD_TSI_ROTARY_ELECTRODE_2  TF_TSI_SELF_CAP_CHANNEL_1      /* PTE4 */
#define FRDM_TOUCH_BOARD_TSI_ROTARY_ELECTRODE_3  TF_TSI_SELF_CAP_CHANNEL_9      /* PTB5 */
#define FRDM_TOUCH_BOARD_TSI_ROTARY_ELECTRODE_4  TF_TSI_SELF_CAP_CHANNEL_8      /* PTB4 */
```

图 26. 软件内 TSI 通道分配

7.4.3 main() 函数中应用代码

```
if ((result = nt_init(&System_0, nt_memory_pool, sizeof(nt_memory_pool))) != NT_S
{
  switch(result)
  {
  case NT_FAILURE:
    nt_printf("\nCannot initialize NXP Touch due to a non-specific error.\n");
    break;
  case NT_OUT_OF_MEMORY:
    nt_printf("\nCannot initialize NXP Touch due to a lack of free memory.\n");
    break;
  }
  while(1); /* add code to handle this error */
}

nt_printf("\nNXP Touch is successfully initialized.\n");

nt_printf("Unused memory: %d bytes, you can make the memory pool smaller without

/* Enable electrodes and controls */
nt_enable();

/* Keypad electrodes*/
nt_control_keypad_set_autorepeat_rate(&Keypad_1, 100, 1000);
nt_control_keypad_register_callback(&Keypad_1, &keypad_callback);

/* Slider electrodes */
nt_control_aslider_register_callback(&ASlider_2, &aslider_callback);

/* Rotary electrodes */
nt_control_arotary_register_callback(&ARotary_3, &arotary_callback);

/* System TSI overflow warning callback */
nt_system_register_callback(&system_callback);

/* Auto TSI register recalibration function, not used as default */
if (recalib_enabled)
{
    recalib_status = (tsi_status_t) nt_module_recalibrate(&nt_tsi_module);
}

if (one_key_only)
nt_control_keypad_only_one_key_valid(&Keypad_1, true);

pit_init();

while(1)
{
  nt_task();

  FMSTR_Poll();
}
}
```


图 27. 应用初始化

7.4.4 软件库同步和处理功能

有两个主要的 API 函数：“nt_trigger()” 和 “nt_task()”。必须由软件应用程序定期调用以触发传感测量并处理结果。

7.4.4.1 nt_trigger 函数

应用程序应在计时器中断或触发任务中定期调用此函数以出发新的数据测量。根据模块的实现，该函数可能会立刻获取数据，或可能仅在中断使能下启用硬件采样。如果触发了 TSI 模块测量，则将扫描整个输入通道序列。这意味着在之前的序列完成之前，不应触发新的序列测量。该函数返回：

- NT_SUCCESS 当执行触发时没有任何错误或警告。
- NT_FAILURE 当检测到出错时，例如模块未准备充分，溢出（数据丢失）错误，等等。不论何种错误，触发都将被初始化。

这是一个 NT 库触发的例子：

```
//For example, there is a callback routine from any periodical source (for example 5 ms)
static void Timer_5msCallBack(void)
{
 if(nt_trigger() != NT_SUCCESS)
  {
    // Trigger error
  }
}
```

图 28. nt_trigger() 函数调用

7.4.4.2 nt_task 函数

应用程序应尽可能频繁地调用此函数，以便处理在数据触发期间获取的数据。每次触发期间应至少调用一次该函数。在内部，该函数将 NT_SYSTEM_MODULE_PROCESS 及 NT_SYSTEM_CONTROL_PROCESS 命令调用传递给在模块和控件配置中的每个对象。该函数返回：

- NT_SUCCESS 当上一次触发获取的数据正在被处理。
- NT_FAILURE 当没有新的数据准备好时。

这是 NT 库中运行任务的示例：

```
// Main never-ending loop of the application
while(1)
{
  if(nt_task() == NT_SUCCESS)
  {
    // New data has been processed
  }
}
```

图 29. nt_task() 函数调用

7.4.5 事件回调函数

每个用户启用的控件都必须为其定义“回调”函数，该函数用于为控件生成的事件（如“ Touch”或“ Release”事件）提供服务。参考下面有关键盘回调的函数。

```
static void keypad_callback(const struct nt_control *control,
                            enum nt_control_keypad_event event,
                            uint32_t index)
{
  switch(event)
  {
  case NT_KEYPAD_RELEASE:

    switch (index) {
        case 0:
            break;
        case 1:
            break;
        case 2:
            break;
        case 3:
            break;
        case 4:
            break;
        case 5:
            break;
        default:
            break;
    }
    break;
  case NT_KEYPAD_TOUCH:

    switch (index) {
        case 0:
            break;
        case 1:
            break;
        case 2:
            break;
        case 3:
            break;
        case 4:
            break;
        case 5:
            break;
        default:
            break;
    }
    break;
```

Release event

Touch event

图 30. 键盘回调函数

7.4.6 nt_setup.c 文件中软件应用的搭建

nt_setup.c 中提供了大多数配置，例如 TSI 寄存器硬件灵敏度，按键检测器设置，电极分配和全局时基。

若定义模块，电极，控件和系统，需要创建结构类型的初始化实例，如以下部分所述。

以下代码展示了 FRDM-KE15z 板上四个电极示例配置 。

恩智浦触摸库中提供了几种按键检测器（触摸评估算法）。电极结构类型必须始终与模块和算法类型匹配。

在软件演示示例中，使用了按键检测器 uSAFA。所有按键可以共享单个按键检测器设置，或者可以将不同的按键检测器设置结构分配给不同的电极，以实现更好的灵活性。

```
struct nt_keydetector_usafa nt_keydetector_usafa_El_1 = {
  .signal_filter.coef1 = 2,
  .base_avrg.n2_order = 9,
  .non_activity_avrg.n2_order = 15,
  .entry_event_cnt = 1,
  .deadband_cnt = 1,
  .signal_to_noise_ratio = 5,
  .min_noise_limit = 200,
  .dc_track_enabled = 0,
  .dc_track_cnt = 100,
};
const struct nt_keydetector_usafa nt_keydetector_usafa_El_3 = {
  .signal_filter.coef1 = 2,
  .base_avrg.n2_order = 12,
  .non_activity_avrg.n2_order = 15,
  .entry_event_cnt = 4,
  .deadband_cnt = 5,
  .signal_to_noise_ratio = 4,
  .min_noise_limit = 60,
  .dc_track_enabled = 1,
  .dc_track_cnt = 100,
};
```

placed in RAM

placed in Flash

图 31. 按键检测器软件定义

电极结构类型必须与应用程序中用于数据测量算法的硬件模块匹配。在本例中，它是 nt_electrode 类型，定义电极参数和 nt_keydetector 接口。

```
/* Electrodes */
const struct nt_electrode El_1 = {
  .shielding_electrode = NULL,
  .keydetector_params.usafa = &nt_keydetector_usafa_El_1,
  .keydetector_interface = &nt_keydetector_usafa_interface,
  .pin_input = FRDM_TOUCH_BOARD_TSI_1,
};
const struct nt_electrode El_2 = {
  .shielding_electrode = NULL,
  .keydetector_params.usafa = &nt_keydetector_usafa_El_1,
  .keydetector_interface = &nt_keydetector_usafa_interface,
  .pin_input = FRDM_TOUCH_BOARD_TSI_2,
};
```

图 32. 电极软件定义

```
/* Modules */
const struct nt_electrode * const nt_tsi_module_electrodes[] = {
  &El_1,  &El_2,  &El_3,  &El_4,  &El_5,  &El_6,  &El_7,  &El_8,  &El_9,  &El_10,
};

const struct nt_module nt_tsi_module = {
  .interface = &nt_module_tsi_interface,
  .wtrmark_hi = 65535,
  .wtrmark_lo = 0,
  .config = (void*)&tsi_hw_config,
  .instance = 0,
  .electrodes = &nt_tsi_module_electrodes[0],
  .safety_interface = &nt_safety_interface,
  .safety_params.gpio = (void*)&my_safety_params,
  .recalib_config = (void*)&recalib_configuration,
};
```

图 33. TSI 模块定义

设置好模块和电极后，即可定义控件。在这种情况下，control_0 是“模拟滑块”控件。

```
const struct nt_electrode * const Keypad_1_controls[] = {
  &El_1,  &El_2,  &El_3,  &El_4,  &El_5,  &El_6, NULL
};
const struct nt_electrode * const ASlider_2_controls[] = {
  &El_7,  &El_8, NULL
};
const struct nt_electrode * const ARotary_3_controls[] = {
  &El_9,  &El_10,  &El_11,  &El_12, NULL
};
const struct nt_control_arotary nt_control_arotary_ARotary_3 = {
  .range = 72,
};
const struct nt_control_aslider nt_control_aslider_ASlider_2 = {
  .range = 160,
  .insensitivity = 2,
};
const struct nt_control_keypad nt_control_keypad_Keypad_1 = {
  .groups = NULL,
  .groups_size = 0,
  .multi_touch = (uint32_t []){0x0C,0x18,0x30,0x24,0x3C,0},
  .multi_touch_size = 5,
};
const struct nt_control Keypad_1 = {
  .electrodes = &Keypad_1_controls[0],
  .control_params.keypad = &nt_control_keypad_Keypad_1,
  .interface = &nt_control_keypad_interface,
};
const struct nt_control ASlider_2 = {
  .electrodes = &ASlider_2_controls[0],
  .control_params.aslider = &nt_control_aslider_ASlider_2,
  .interface = &nt_control_aslider_interface,
};
const struct nt_control ARotary_3 = {
  .electrodes = &ARotary_3_controls[0],
  .control_params.arotary = &nt_control_arotary_ARotary_3,
  .interface = &nt_control_arotary_interface,
};
```

Keypad electrodes

Slider electrodes

Rotary electrodes

Rotary range

Slider range and granularity

图 34. 键盘控件定义示例

现在，我们准备将所有部分连接到“系统”结构中。

```
/* System */
const struct nt_control * const System_0_controls[] = {
  &Keypad_1,  &ASlider_2,  &ARotary_3, NULL
};
const struct nt_module * const System_0_modules[] = {
  &nt_tsi_module, NULL
};
const struct nt_system System_0 = {
  .time_period = TIME_PERIOD,
  .init_time = 400,
  .safety_period_multiple = 0,
  .safety_crc_hw = true,
  .controls = &System_0_controls[0],
  .modules = &System_0_modules[0],
};
```

图 35. nt_system 软件配置

Kinetis E 系列 MCU 包含最先进的 TSI v5 外设。必须正确地将模块进行配置。但是，NT 库在应用程序开发过程中会有所帮助，并且不必处理 TSI 模块的差异。TSI 硬件设置如 图36 所示。“ tsi_hw_config” 包含用于自电容模式和互电容模式的寄存器设置。

如果应用程序未使用这两个模块，则略冗余寄存器的设置将被忽略。

```
const  tsi_config_t tsi_hw_config = {
  .configSelfCap.commonConfig.mainClock = kTSI_MainClockSlection_0,
  .configSelfCap.commonConfig.ssc_mode = kTSI_ssc_prbs_method,
  .configSelfCap.commonConfig.mode = kTSI_SensingModeSlection_Self,
  .configSelfCap.commonConfig.dvolt = kTSI_DvoltOption_3,
  .configSelfCap.commonConfig.cutoff = kTSI_SincCutoffDiv_0,
  .configSelfCap.commonConfig.order = kTSI_SincFilterOrder_2,
  .configSelfCap.commonConfig.decimation = kTSI_SincDecimationValue_4,
  .configSelfCap.commonConfig.chargeNum = kTSI_SscChargeNumValue_4,
  .configSelfCap.commonConfig.prbsOutsel = kTSI_SscPrbsOutsel_2,
  .configSelfCap.commonConfig.noChargeNum = kTSI_SscNoChargeNumValue_2,
  .configSelfCap.commonConfig.ssc_prescaler = kTSI_ssc_div_by_2,
  .configSelfCap.enableSensitivity = true,
  .configSelfCap.enableShield = false,
  .configSelfCap.xdn = kTSI_SensitivityXdnOption_3,
  .configSelfCap.ctrim = kTSI_SensitivityCtrimOption_0,
  .configSelfCap.inputCurrent = kTSI_CurrentMultipleInputValue_0,
  .configSelfCap.chargeCurrent = kTSI_CurrentMultipleChargeValue_0,
  .configMutual.commonConfig.mainClock = kTSI_MainClockSlection_0,
  .configMutual.commonConfig.ssc_mode = kTSI_ssc_prbs_method,
  .configMutual.commonConfig.mode = kTSI_SensingModeSlection_Mutual,
  .configMutual.commonConfig.dvolt = kTSI_DvoltOption_3,
  .configMutual.commonConfig.cutoff = kTSI_SincCutoffDiv_0,
  .configMutual.commonConfig.order = kTSI_SincFilterOrder_2,
  .configMutual.commonConfig.decimation = kTSI_SincDecimationValue_4,
  .configMutual.commonConfig.chargeNum = kTSI_SscChargeNumValue_4,
  .configMutual.commonConfig.noChargeNum = kTSI_SscNoChargeNumValue_2,
  .configMutual.commonConfig.prbsOutsel = kTSI_SscPrbsOutsel_2,
  .configMutual.commonConfig.ssc_prescaler = kTSI_ssc_div_by_2,
  .configMutual.preCurrent = kTSI_MutualPreCurrent_4uA,
  .configMutual.preResistor = kTSI_MutualPreResistor_4k,
  .configMutual.senseResistor = kTSI_MutualSenseResistor_10k,
  .configMutual.boostCurrent = kTSI_MutualSenseBoostCurrent_0uA,
  .configMutual.txDriveMode = kTSI_MutualTxDriveModeOption_0,
  .configMutual.pmosLeftCurrent = kTSI_MutualPmosCurrentMirrorLeft_32,
  .configMutual.pmosRightCurrent = kTSI_MutualPmosCurrentMirrorRight_1,
  .configMutual.enableNmosMirror = true,
  .configMutual.nmosCurrent = kTSI_MutualNmosCurrentMirror_1,
  .thresl = 0,
  .thresh = 65535,
};
```

图 36. 软件中 TSI 硬件设置

7.5 恩智浦触摸库内存要求

内存需求取决于“应用程序的大小”，从根本上说取决于电极输入的数量以及应用程序中使用的控件（如键盘按键，滑块等）的数量。

Touch 库是面向 32 位 Kinetis Arm cortex-M MCU 新编写的。为了利用 32 位结构的优势，我们将大多数结构，指针和常量以 32 位格式存储在 Flash 中。RAM 中几乎没有放置任何内容。

由于这种方法，Flash 中的数据可以正确对齐，并且具有适合 Arm 计算的大小，而无需 CPU 进一步的位操作，并且不会损失精度。这样可以节省计算过程中的 CPU 时间。

这也防止了与移植到不同编译器的有关问题。

最小的 Kinetis-L 设备具有 16 kB Flash，而大多数支持 TSI 外设的设备具有 32 kB 或更多的片上 Flash。可以软件的触摸感应部分安装到 32 kB Flash Kinetis 器件中，并为其余应用保留空间。

请参阅下表中关于在 KSDK（包括）上传递软件项目以及禁用的 FreeMASTER 的典型内存要求。

大多数 Flash（ROM）常量是在 “nt_setup.c” 配置中包含的结构中定义的文件。通过此文件的大小和复杂性，我们可以估计总共的内存需求。

7.5.1 内存空间优化

所需的所有 Flash 空间存取决于触摸应用程序的复杂性，比如启用的电极和控件的数量。随着复杂性的增加，所需的 Flash 和 RAM 的大小也成比例地增加。 通过删除结构定义之前的“ const”关键字，可以将关键检测器 C 结构临时放置到 RAM 中，请参见图 29。 电极数据、键检测器和过滤器计算所使用的大多数运行变量都是在 “nt_memory_pool[]” 上创建的，它被定义为一个静态 RAM 数组，其大小完全根据应用程序的需要所选择。参考 “nt_memory_pool” 定义为 4000 字节大小的数组。请注意，根据编译器的不同，ARM Cortex-M 内核需要正确对齐。可以使用 “nt_mem_get_free_size” 来估计数组大小，该值在正确初始化后返回剩余内存。

```
#if defined(__ICCARM__)
  uint8_t nt_memory_pool[4000]; /* IAR EWARM compiler */
#else
  uint8_t nt_memory_pool[4000] __attribute__((aligned (4)));  /* Keil, GCC compi
#endif
```

图 37. nt_memory_pool 初始化

```
if ((result = nt_init(&System_0, nt_memory_pool, sizeof(nt_memory_pool))) != NT_SUCCESS)
{
  switch(result)
  {
  case NT_FAILURE:
    nt_printf("\nCannot initialize NXP Touch due to a non-specific error.\n");
    break;
  case NT_OUT_OF_MEMORY:
    nt_printf("\nCannot initialize NXP Touch due to a lack of free memory.\n");
    break;
  }
  while(1); /* add code to handle this error */
}

nt_printf("\nNXP Touch is successfully initialized.\n");
nt_printf("Unused memory: %d bytes \n", (int)nt_mem_get_free_size());
```

图 38. Nt_memory_pool 大小

7.5.2 移除 FreeMASTER

默认情况下，FreeMASTER 在触摸演示例程中被使能，这是在开发阶段运行 FreeMASTER GUI 所必需的。FreeMASTER 本身会消耗一些资源，因此建议在触摸感应调整完成后立即将其从最终的软件项目中删除。FreeMASTER 支持可以通过定义 “NT\U FreeMASTER\U support 0” 在触摸应用程序中全局禁用，请参见 图 39。

```
#ifndef NT_FREEMASTER_SUPPORT
  #define NT_FREEMASTER_SUPPORT 0
#endif
```

图 39. 在触摸库中禁用 FreeMASTER 支持

FreeMASTSTER 剩余的定义和引用，比如 TSA 表，“init_freemaster_uart()” 和“FMSTR_Init()”函数必须从 SW 项目中移除，请参见 图 40。

```
/*
 * This list describes all TSA tables that should be exported to the
 * FreeMASTER application.
 */
//#ifndef FMSTR_PE_USED
//  FMSTR_TSA_TABLE_LIST_BEGIN()
//      FMSTR_TSA_TABLE(nt_frmstr_tsa_table)
//  FMSTR_TSA_TABLE_LIST_END()
//#endif



 /* FreeMASTER communicates over the default UART instance */
 //init_freemaster_uart();

 /* FreeMASTER initialization */
 //(void)FMSTR_Init();

 while(1)
 {
   nt_task();

   //FMSTR_Poll();
 }
```

图 40. 移除 FreeMASTER 参考

表 1. 恩智浦触摸通用应用要求（FreeMASTER 已移除）

App. 大小	2 电极	28 电极
FLASH [kB]	14	28
SRAM [kB]	2.2	7.2

8 按键探测器 uSAFA

恩智浦触摸库支持三种按键检测器：

- AFID — 专利已授权，简单的 CPU 计算
- SAFA — 自适应滤波算法（专利已授权）
- uSAFA — “单向” SAFA（推荐）

SAFA 是触摸软件库使用过的最先进的算法。

SAFA 即信号自适应滤波算法，是 NXP 申请专利授权的一种滤波软件算法，这里的“u”表示单向。它是基于不同权值的移动平均滤波器。噪声级（死区）限制与噪声级跟踪和自动阈值自适应一起使用。典型的触摸信号高低被跟踪并用作“预测”的触摸信号值。

8.1 按键检测器 uSAFA 相关信号

以下信号最为重要：

- **基线**是基本的参考信号，它随着时间和环境的变化而缓慢移动，是其他所有信号的参考。

- **信号电平**是基本过滤的原始触摸感应信号（使用软件低通滤波器）。该信号反映变化如果触摸的话。
- 噪声下限（**最小噪声限制**），我们预计在正常环境中环境系统噪声不会超过此值（无附加 EMC 噪声）。这意味着通常系统噪声远低于此值。我们在实验中将它设置为 **100**，而实际值可能要低得多。
- **死区**是噪声电平，信号必须穿过死区才能检测触摸或释放状态。在这种情况下，死区定义为（最小噪声极限 x 信噪比）。例如，当 SNR=6 时，则**死区=100*6=600** 计数。
- **预测信号**是触摸按钮时的典型信号电平。我们在触摸或释放时调整此值。
- 对于**触摸**事件，信号必须高于预测信号的 25%，并越过死区电平。
- 对于**释放**事件，信号必须低于预测信号的 80%。
- 事件计数器（“entry_event_cnt” 及 “deadband_cnt”）被用于消除触摸和释放事件的抖动。

```
/* SAFA keydetector settings */
const struct nt_keydetector_usafa  keydec_usafa =
{
/* Electrodes */
    .signal_filter = {2},
    .base_avrg  = {.n2_order  = 10},
    .non_activity_avrg  =  {.n2_order  = NT_FILTER_MOVING_AVERAGE_MAX_ORDER},
    .entry_event_cnt  = 4,
    .deadband_cnt  = 4,
    .signal_to_noise_ratio = 6,
    .min_noise_limit = 100,
    .dc_track_enabled = 1,
    .dc_track_cnt = 100,
};
```

图 41. uSAFA 死区阈值设置

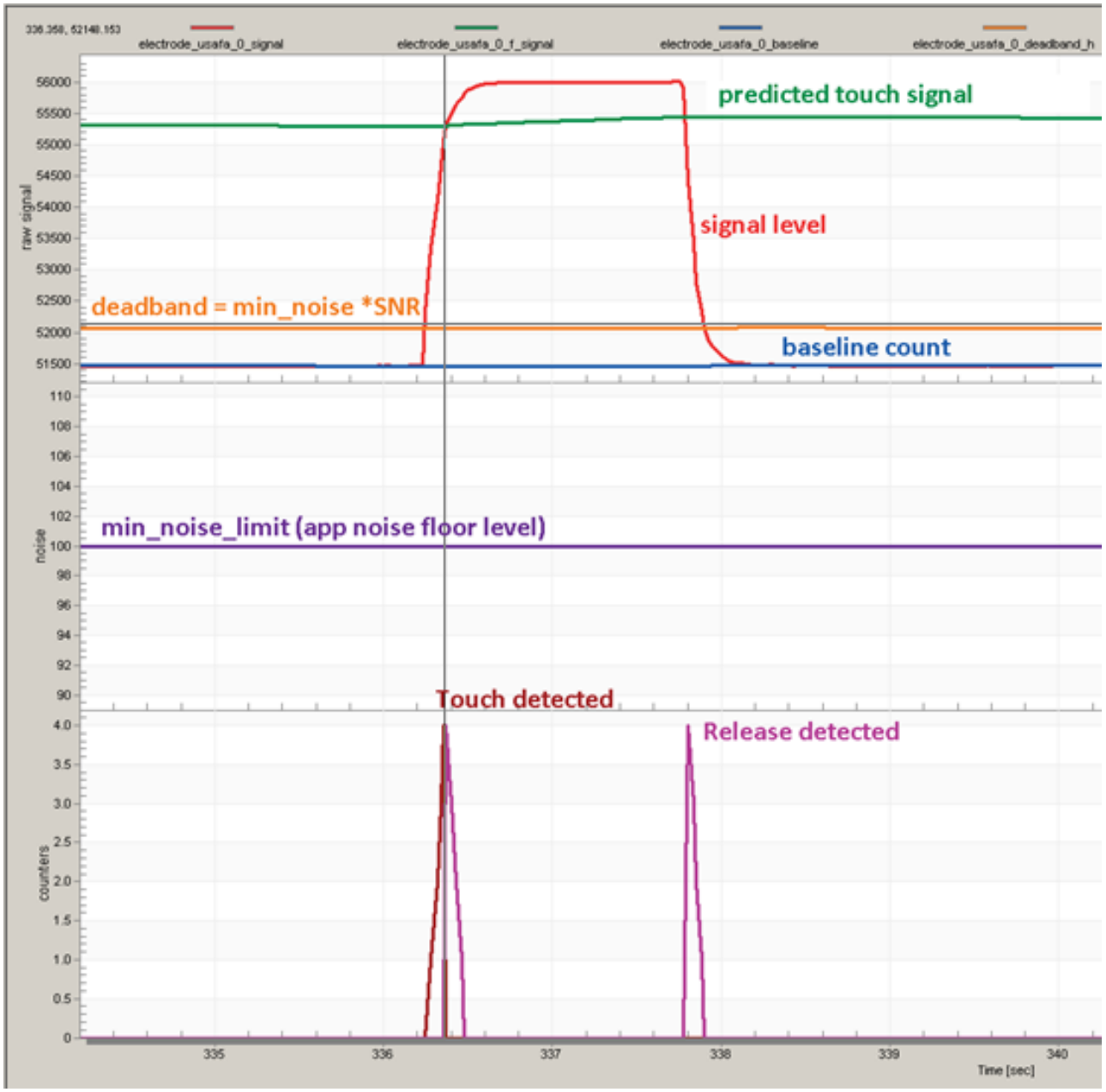


图 42. 按键检测器 uSAFA 信号

8.2 按键检测器 uSAFA 滤波参数

所有滤波器均基于移动软件滤波，其阶数取二的指数（2^n）。

比如，阶数 = 10 意味着， 2 的 10 次方 = 1024 个样本会被取均值。

这意味着如果我们增加到 11，2048 个样本会被平均过滤速度将慢两倍。

相反，如果我们减少到 9 或 8，它会快 2 到 4 倍。因此通过改变滤波器的阶数，可以控制响应和自适应速度。所有过滤器和去抖动计数器都取决于 time_period 这个参数，其值等于 TSI 扫描周期。

请参见 图 43 以及描述参数的注释：

```
/* SAFA keydetector settings */
const struct nt_keydetector_usafa keydec_usafa =
{
.signal_filter = {2},                  // Coefficient of the input IIR signal filter, used to suppress high-frequency noise.
   .base_avrg = {.n2_order = 10}, // Settings of the moving average filter for the baseline in the release state of an electrode.
   .non_activity_avrg =  {.n2_order = 15}, // Settings of the moving average filter for the signals in the inactivity state of an electrode. (for example baseline in a touch state).
   .entry_event_cnt = 4, // Sample count for the touch event. This means that this count of samples must meet the touch condition to trigger a real touch event.
   .deadband_cnt = 4,    // Sample count for the deadband filter. This field specifies the number of samples that cannot proceed to the next event.
   .signal_to_noise_ratio = 6, // Signal-to-noise ratio – it is used for counting the minimum size of the signal that is ignored
   .min_noise_limit = 100,     // Minimal system noise floor level
   .dc_track_enabled = 1,      // When the DC-tracker is enabled, then the Keydetector resets, when there is a significant signal drop below the baseline detected
   .dc_track_cnt = 100,        // Number of samples requested (length of the negative signal drop) after that the DC tracker resets the keydetector.
};

const struct nt_system system_0 = {
   .controls = &controls[0],
   .modules = &modules[0],
   .time_period = 50, // Time base [ms] used for sampling the electrodes (HW timer trigger firing the sequence)  and to measure DC tracker time-out.
   .init_time = 400,  // Application warm-up delay period [ms], while the signals can settle but aren't evaluated
};

LPIT_SetTimerPeriod(LPIT0, kLPIT_Chnl_0, 50*CLOCK_GetFreq(kCLOCK_ScgSircAsyncDiv2Clk)/1000); // Configure HW timer period in "main.c" properly
```

图 43. 按键检测器 uSAFA 参数

8.3 直流跟踪器功能

直流跟踪器有助于适应信号突然下降到基线以下的特殊情况。这可能真实地发生在应用程序中。

例如，当在通电期间传感器电极上存在一些物体，并且在一段时间后将其移除时，它必须恢复并调整阈值以适应新的情况，以便能够检测到“常规”接触。

直流跟踪器复位按键检测器当信号下降高于（2 * min_noise_limit）。直流跟踪器的反应时间是可设置的，并由多个 dc_track_cnt * time_period 所确定。在下面的示例中，按键检测器在超时 5000 ms 复位。

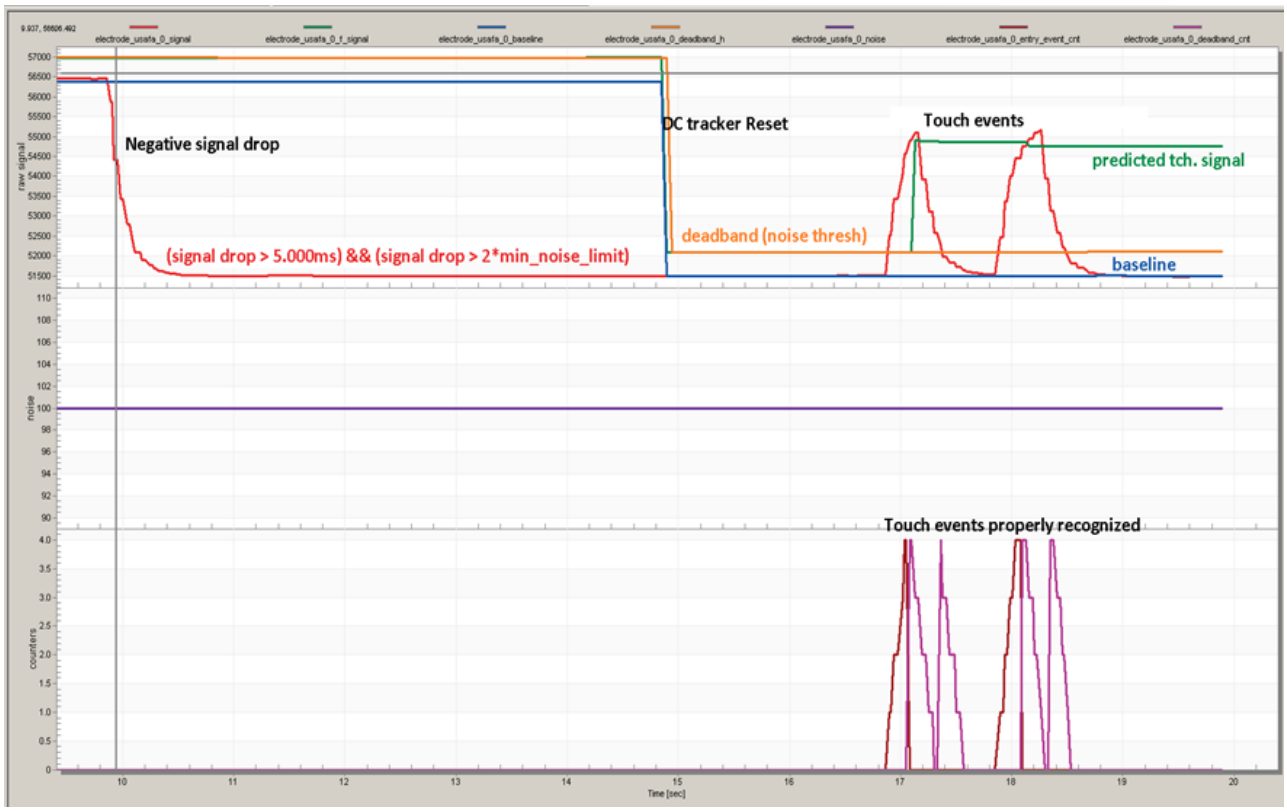


图 44. 直流追踪器反应

8.4 按键检测器 uSAFA 调制

通过改变按键检测器的参数，可以调整触摸敏感阈值，改变噪声自适应速度以及对外界影响的鲁棒性。

如果我们将 SNR 从 6 改至 15, 死区阈值将从 600 变为 1500。

- 忽略小于 min_noise_limit (100) 的增量信号值，将其作为噪声界值。
- 增量信号值大于 min_noise_limit (100) ，而小于死区，则被作为增加的系统噪声，用于自动死区阈值自适应。
- 增量值大于死区被当作首要触摸事件处理。
- 次要触摸事件是超过预期触摸信号的 25 %。
- 必须满足这两个条件才能触发触摸事件计数器。
- 如果触摸事件数 ≥ “entry_event_cnt”（用来消除小故障），则触摸有软件进行评估。

```
const struct nt_keydetector_usafa keydec_usafa =
{
.signal_filter = {2},
    .base_avrg = {.n2_order = 10},
    .non_activity_avrg =  {.n2_order = 15},
    .entry_event_cnt = 4,
    .deadband_cnt = 4,
    .signal_to_noise_ratio = 15, // 6
    .min_noise_limit = 100,
    .dc_track_enabled = 1,
    .dc_track_cnt = 100,
};
```

图 45. uSAFA 按键检测器设置

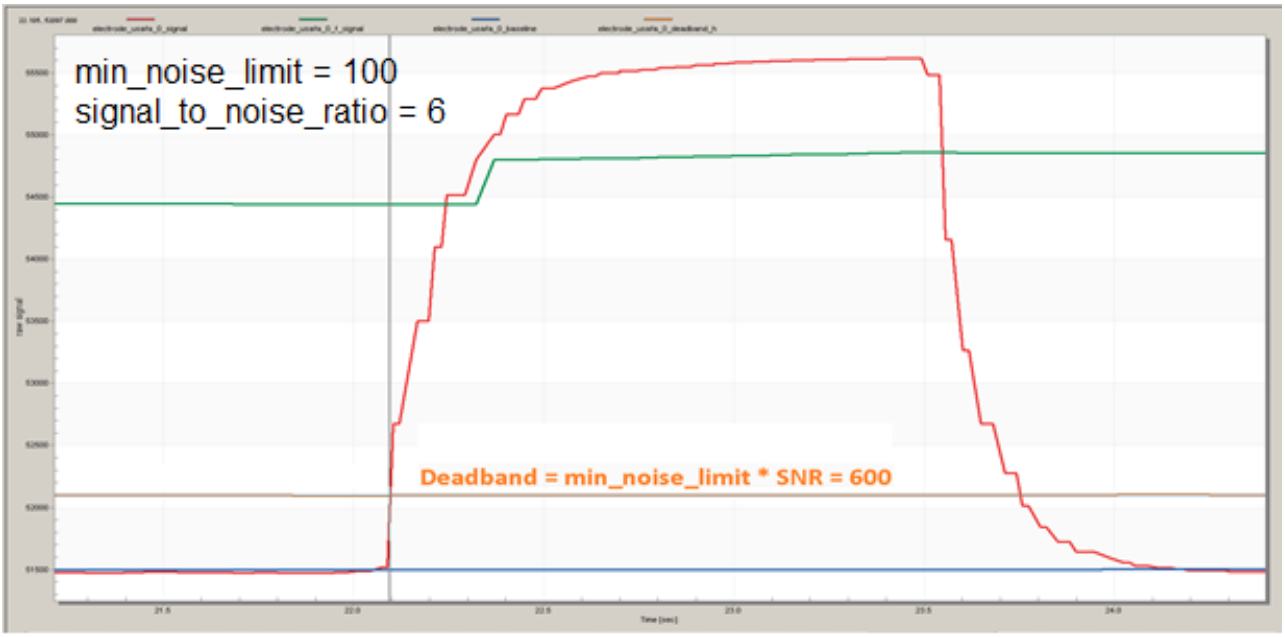


图 46. uSAFA 调制 SNR = 6

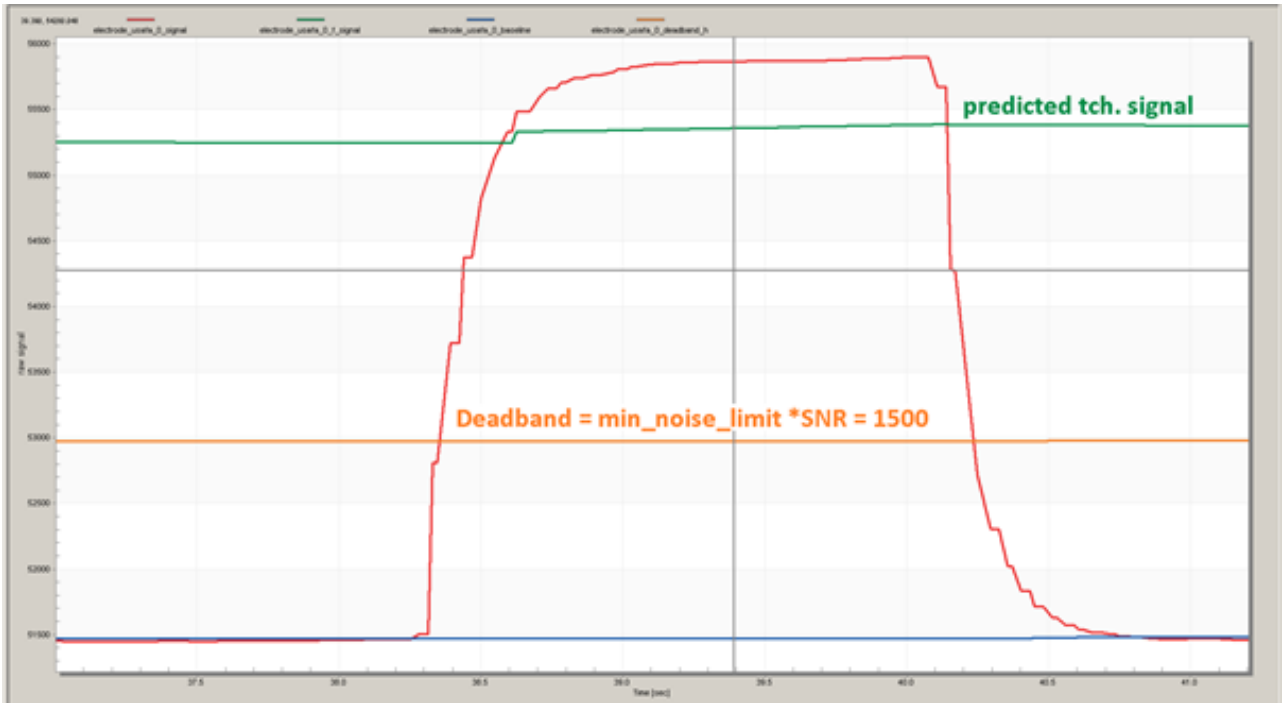


图 47. uSAFA 调制 SNR = 15

8.4.1 噪声电平自适应

噪声检测和噪声电平自适应用于克服恶劣环境或通过 EMC 工作台上的抗扰度测试。

若噪声信号超过了 min_noise_level (100)，则噪声电平将被累积和增加，使死区被更新为 noise_level * SNR。

例如，如果噪声电平上升到 130，则死区从（100*15）更新到（130*15），使得阈值上升 130%，以适应增加的噪声条件。

当噪声电平下降时，它会在一段时间后自动下降至 min_noise_level (100)。噪声电平自适应和恢复的速度由软件滤波器 base_avrg 的阶数控制。

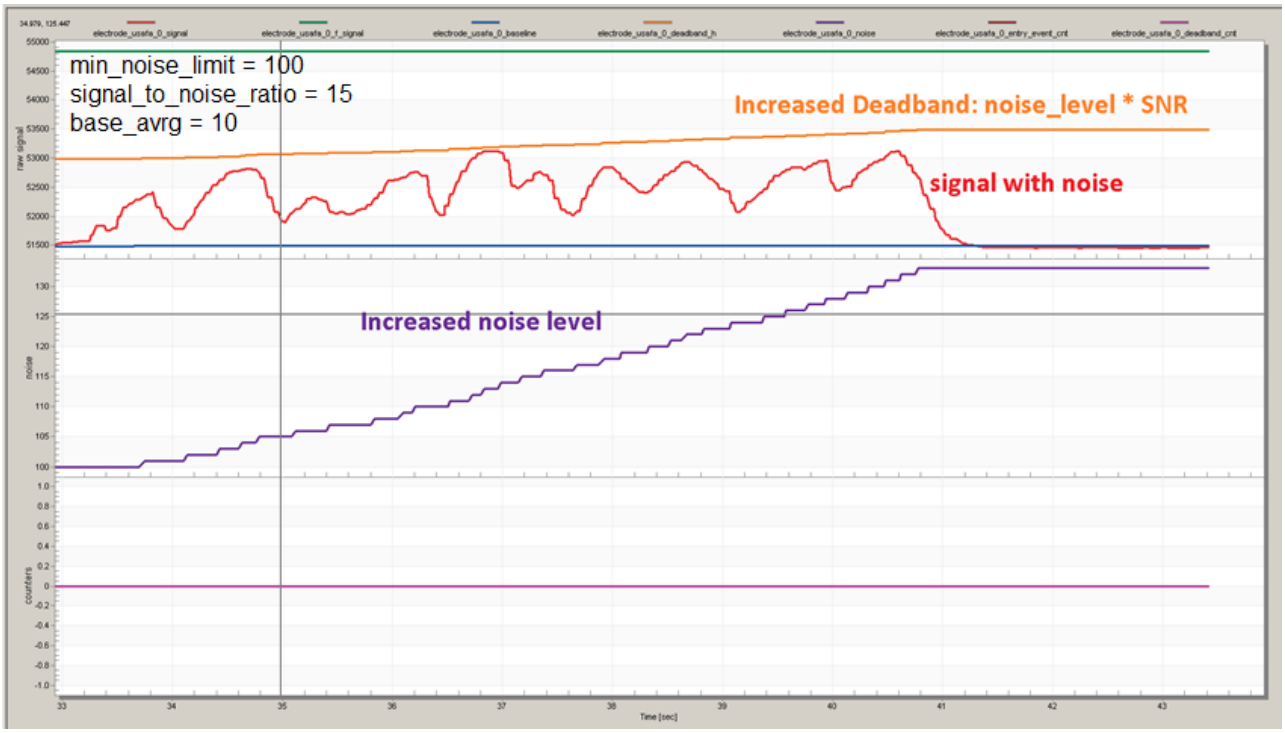


图 48. 噪声电平自适应

如果我们将 base_avrg 滤波器阶数由 10 改为 8， 噪声级自适应将比以前快 4 倍，这样它可以更快地更新死区阈值并对增加的噪声作出反应。请注意，当噪声信号消失时，噪声电平会自动恢复。噪声电平恢复速度被硬编码为比噪声累积速度快 16 倍，这适用于大多数情况，但如果有需要的话，也可在软件中进行加速。

```
const struct nt_keydetector_usafa keydec_usafa =
{
.signal_filter = {2},
   .base_avrg = {.n2_order = 8}, //10
    .non_activity_avrg =  {.n2_order = 15},
    .entry_event_cnt = 4,
    .deadband_cnt = 4,
    .signal_to_noise_ratio = 15, // 6
    .min_noise_limit = 100,
    .dc_track_enabled = 1,
    .dc_track_cnt = 100,
};
```

图 49. uSAFA 噪声自适应的基本滤波器调制

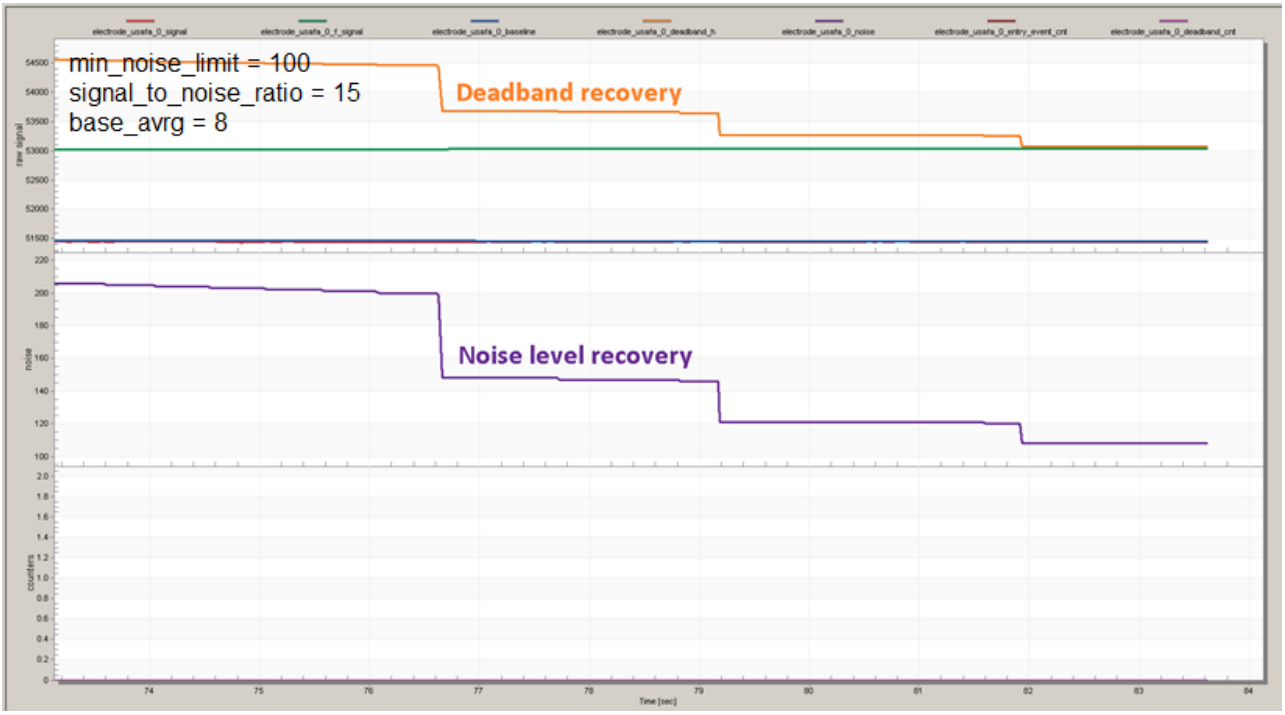


图 50. 噪声电平恢复

9 TSI 模块硬件介绍

9.1 TSI v5 主要特征

- 支持自感式传感器和互感式传感器
- 增强了对系统 EMC 标准测试的抗扰性
- 增强的灵敏度以支持不同的面板厚度
- 能够将 MCU 从 Stop2 和低功耗模式唤醒
- 完全支持恩智浦触摸感应软件（NT）库
- 支持 DMA 数据传输

9.2 TSI 方法

触摸感应接口（TSI）在电容式触摸传感器上提供触摸感应检测。外部电容式触摸传感器常在 PCB 上形成，并且传感器电极通过设备中的 I/O 引脚连接到 TSI 输入通道。

支持两种不同的触摸感应方法，自电容模式和互电容模式。

KE15z MCU 在自电容模式下最多支持 25 个输入，并且可以在互模式下实现 6 × 6 输入。两种方法都可以在单个 PCB 上组合，只能将较低的 12 个 TSI 通道 TSI[0:11] 用于相互模式。请注意，在互感模式下，TSI[0:5] 是 TSI Tx 引脚，TSI[6:11] 是 TSI Rx 引脚。

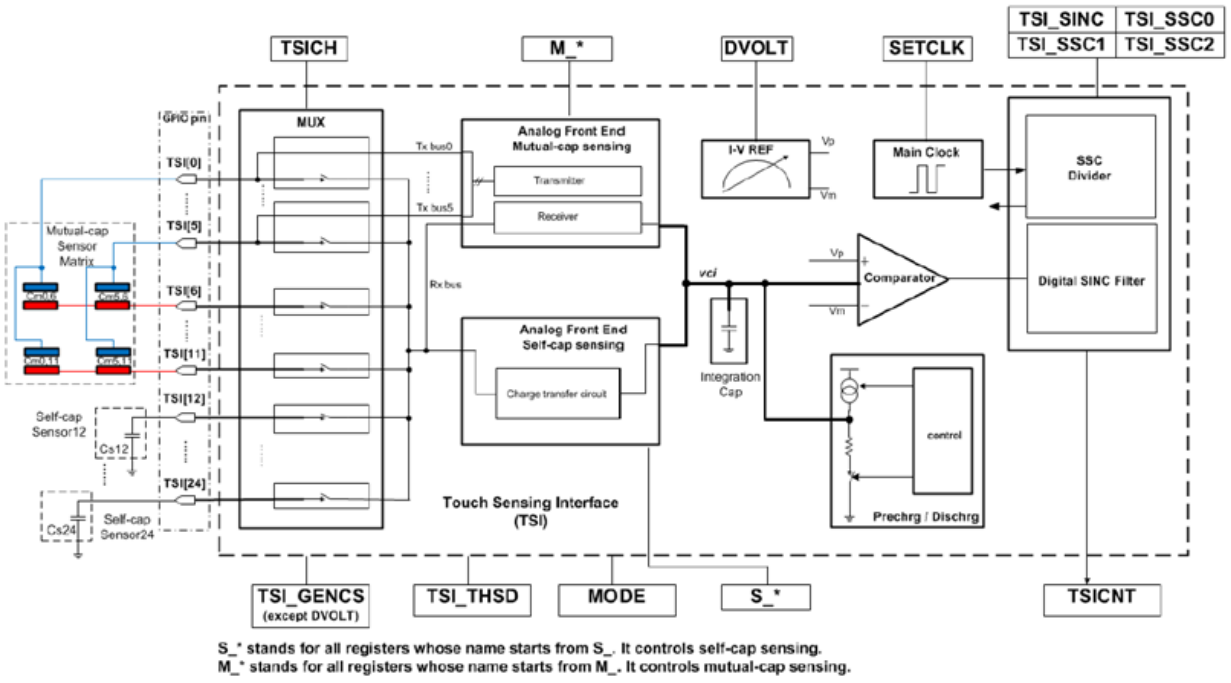


图 51. Kinetis KE1x TSI v5 框图

9.3 自电容和互电容

自电容传感器和互电容传感器之间的电场分布不同。

对于自电容，电极与系统接地之间存在电容。通过人体触摸变化场并产生额外的电容。

对于互电容，两个电极之间存在感测电容。通过人体触摸变化场并减少相互电容。TSI IP 可以将电容从传感器转换为数字代码以进行应用。

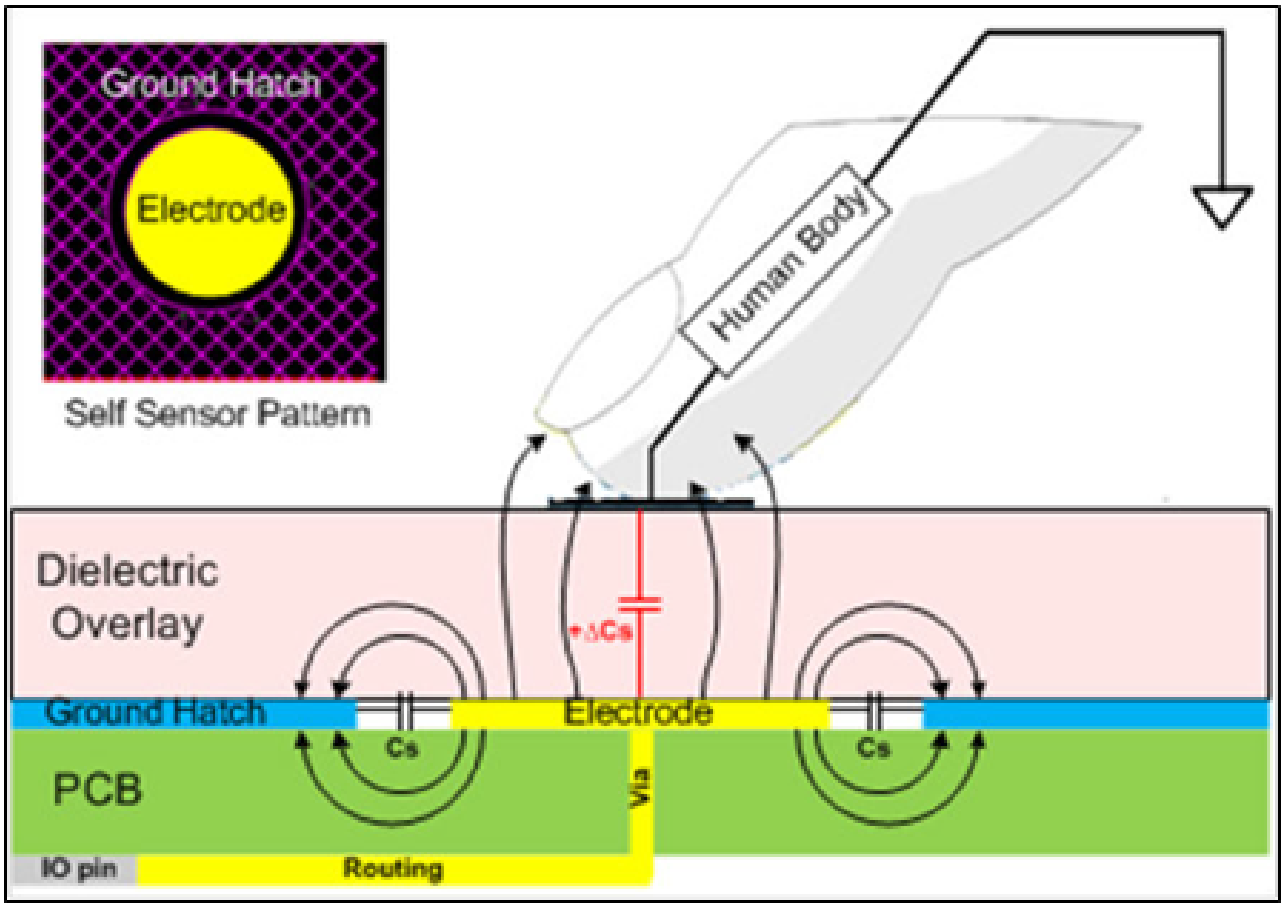

Ground Hatch
Electrode
Self Sensor Pattern
Human Body
Dielectric
Overlay
+ΔCs
Ground Hatch
Electrode
Cs
Cs
PCB
Via
IO pin
Routing

图 52. 自电容原理

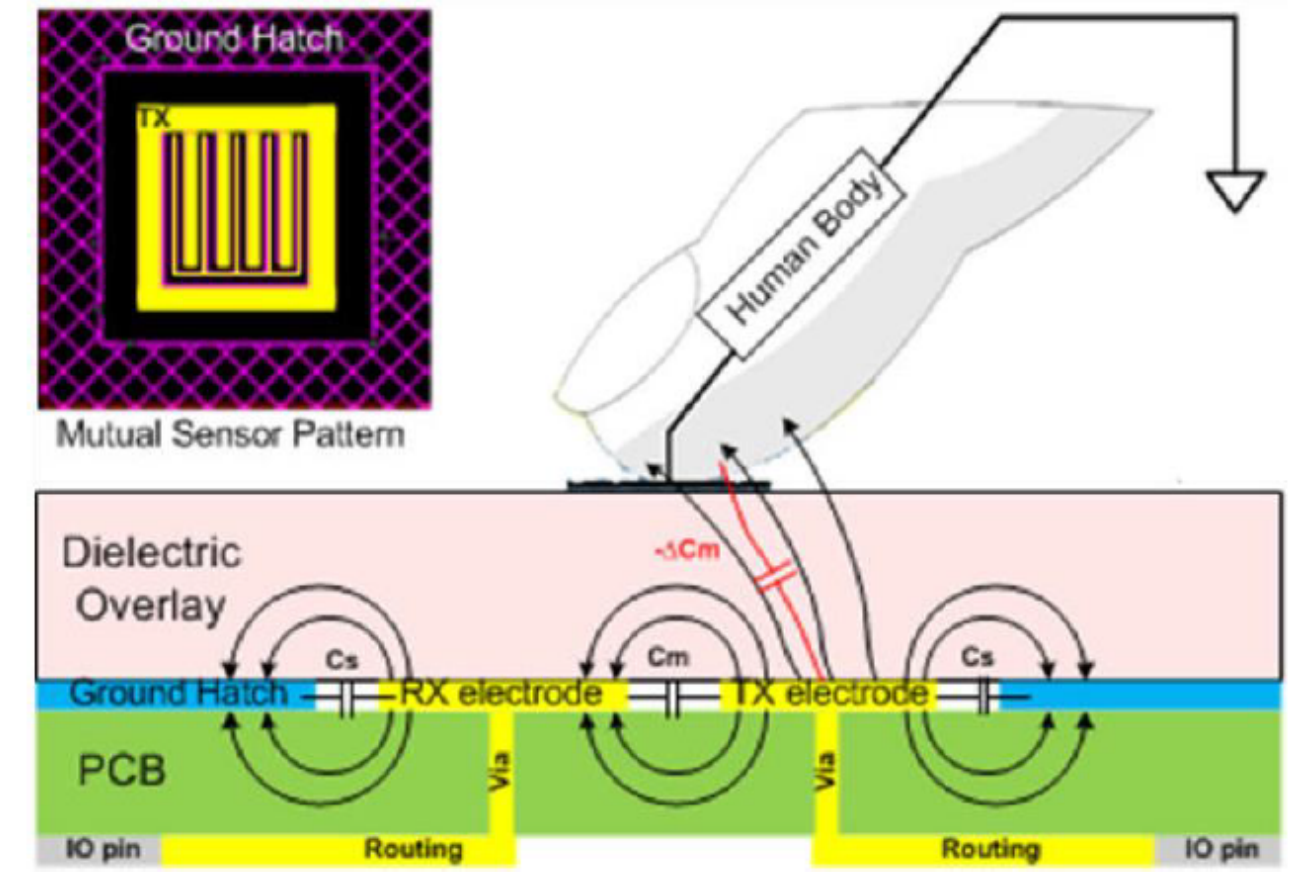


图 53. 互电容原理

9.4 自感式硬件架构

电荷转移方法（具有固有的噪声免疫性）用于检测触摸事件。控制一个包括不重叠的 ph1（采样阶段）和 ph2（转移阶段）的采样时钟以对电极电容器充电，并通过电荷转移电路（CTC）将电荷转移到内部集成电容器。阶梯锯齿在 V_{ci} 节点处生成。比较器检测到 V_{ci}，当它超过正参考 V_p 时，C_i 将放电到负参考 V_m。然后继续下一个扫描周期。触摸发生时，输入电容将增加，然后减少锯齿上升的步数。通过数字滤波器检测数量差异。数字滤波器抑制了数量和输出计数的噪声，软件可以使用这些噪声来检测触摸。

恩智浦触摸软件库在软件中执行触摸信号转换。

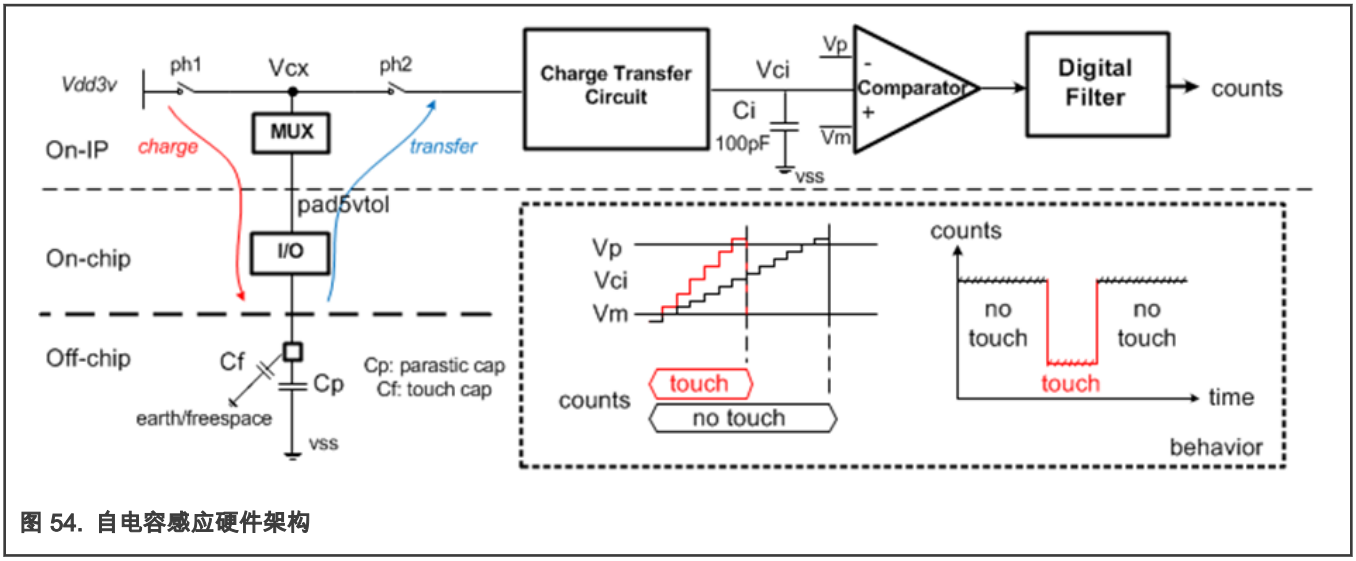


图 54. 自电容感应硬件架构

9.5 互感式硬件架构

互电容感应方式包括发送通道和接收通道。在时钟控制下，发送通道输出脉冲，这些脉冲通过互电容耦合到接收通道位置。接收通道采用类似于自电容感应中的电荷转移电路，放大信号并且消除了噪声。

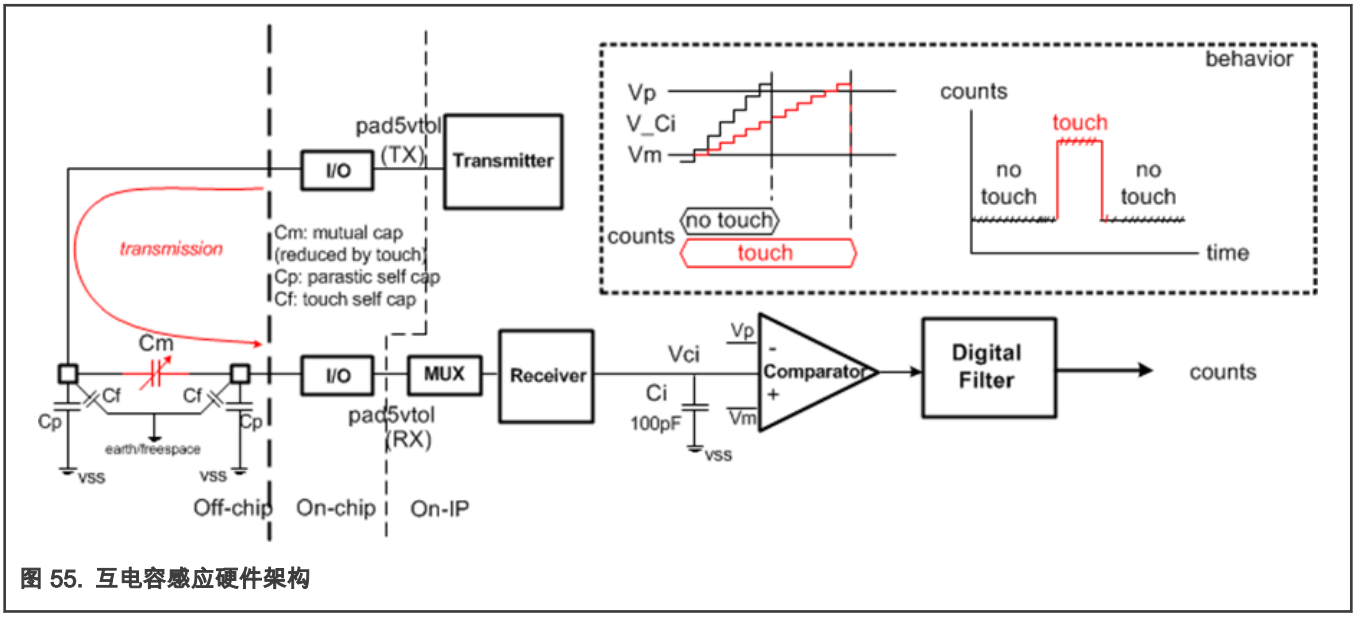


图 55. 互电容感应硬件架构

9.6 了解 TSI 测量

9.6.1 自电容模式的详细信息

在 TSI IP 模块内部，通过不重叠的时钟 ph1/ph2 和跨导放大器来操作 TSI 扫描。TSI 扫描模块分别由 ph1 和 ph2 控制两个阶段：

- 采样阶段：当 ph1 打开时，开关 ph1 控制采样阶段，外部触摸电极 $_x$ 由 vdd3v 充电。
- 充电阶段：开关 ph2 控制充电阶段，当 ph1 关断然后 ph2 导通时，电容器 C_x 上的充电流向内部集成电容器 C_i，从而产生平均电流 I_{cx}。

通过由两个电流镜组成的跨导放大器，还根据输入和充电电流镜设置来放大 I_{cx}。最终平均电流，将积分的 C_i 充电为 $X_{ch}*X_{in}*I_{cx}$。由于集成的 C_i 是通过平均电流充电的，电压 V_{ci} 会在 C_i 上积累，因此当 V_{ci} 大于预设的 V_p 时，比较器将停止对此 TSI 进行扫描并将扫描结果记录到 TSICNT 寄存器中。

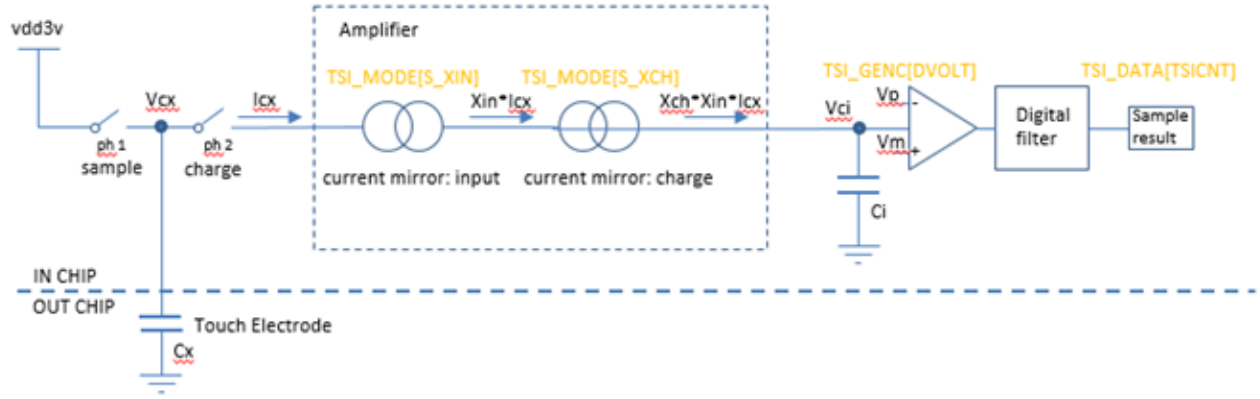


图 56. 自电容模块

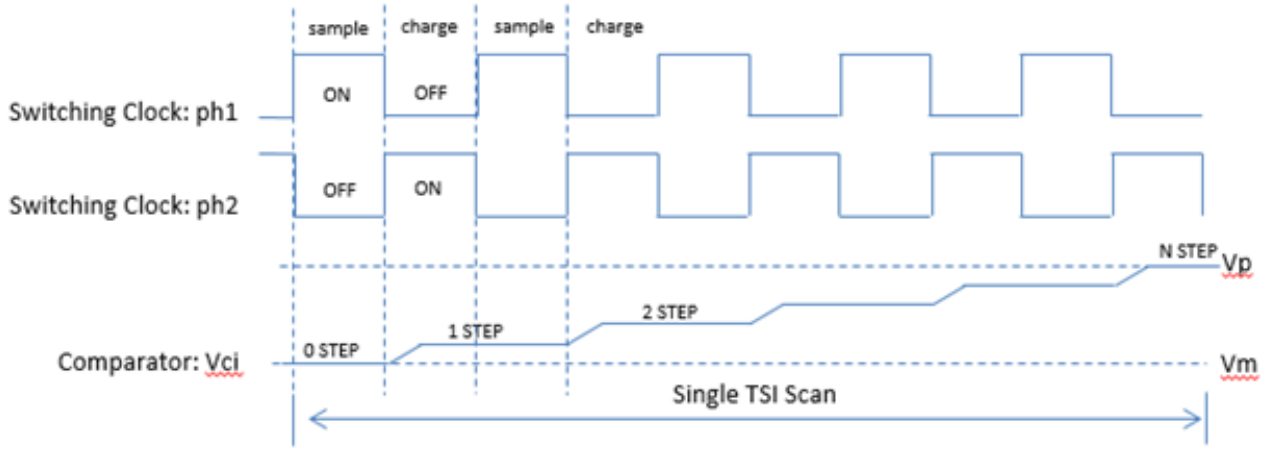


图 57. 自电容模式时序

9.6.2 互电容模式的详细信息

互电容模式用于测量连接到两个 TSI 通道的两个电极之间的电容。TSI 通道之一用作发送（TX）通道，另一通道用于接受（RX）通道。

对于 TSI 互电容模式，有两个阶段由开关时钟控制：

- 充电阶段：开关 ph1 控制充电阶段，当 ph1 接通时，传输通道输出脉冲通过互电容 C_m 耦合。接收通道将接收到的电压脉冲（$V_{pre}+\Delta V$）转换为通过电阻 R_s 的电流 I_{charge}。
- 放电阶段：开关 ph2 控制放电阶段，当 ph1 关闭然后 ph2 打开时，传输通道将电压从 Vdd5V 改为 0 V。接收通道通过 R_s 将接收到的电压变化（Vpre+ΔV）转换为电流 $I_{discharge}$。

由于通过接收通道的镜像/放大电流对对集成 C_i 进行充电/放电，因此当 V_{ci} 大于预设 V_p 时，C_i 上的电压 V_{ci} 斜坡将上升，比较器将对此数据进行扫描并记录下来。样本结果为 TSICNT。

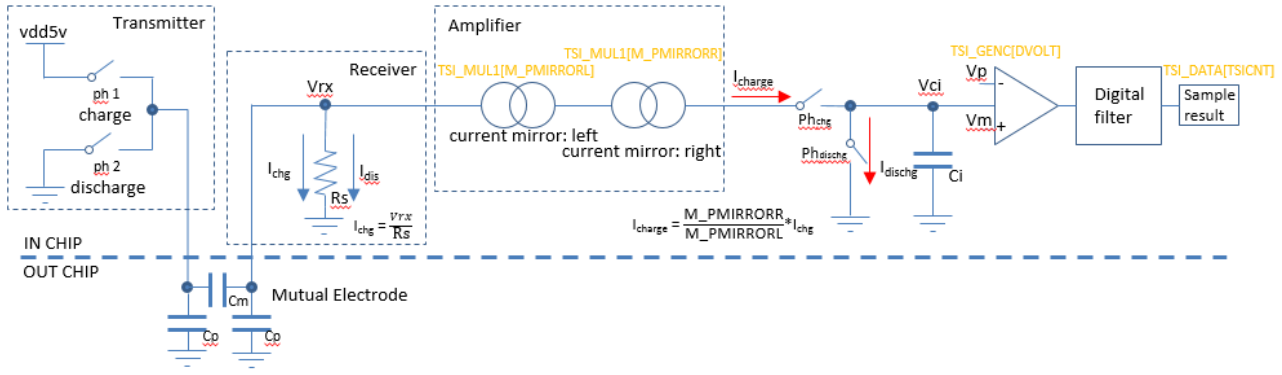


图 58. 互电容模块

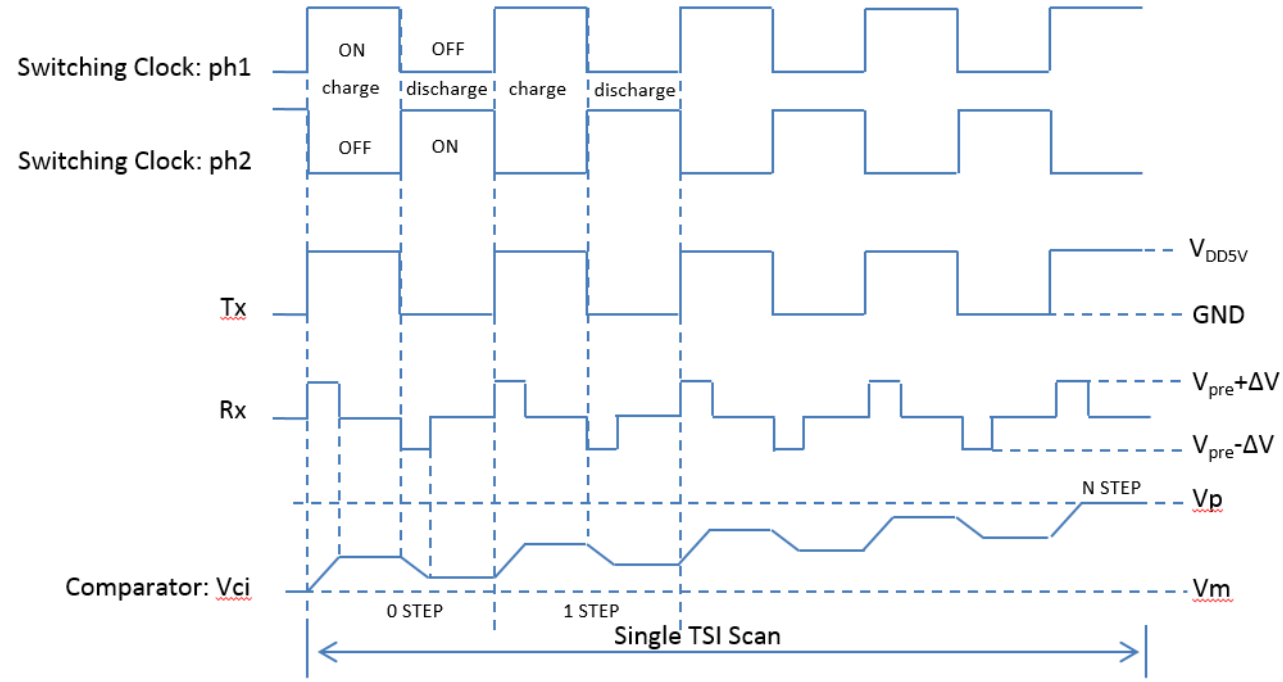


图 59. 互电容模式时序

注意

由于早期 MKE15z256 芯片具有硬件限制，如果 TSI 在互操作模式下运行，则将保留 TSI 通道 0 至 5，用于 TSI 功能，并且不能用于 GPIO 等其他用途。在设计电路板是必须考虑到这一点。该硬件限制已在更高版本的 KE16x 设备上修复。

9.7 TSI IP 硬件寄存器调整 — 自电容模式

硬件寄存器设置可在软件结构中找到：hw_config 位于文件 nt_setup.c 中。该结构包括自电容和互电容模式的可调参数。

在自电容模式下，基本模式是禁用灵敏度增强功能。启用灵敏度增强后，由于可以使用更多参数，因此调整变得更加灵活。

9.7.1 禁用增强时的自电容灵敏度

在这种模式下，由于一些参数的调整变得更加容易，这些参数对于灵敏度和累积时间很重要。有关以绿色突出显示的重要参数的更多详细信息，请参见 图 60。

注意

NSTEP 是 TSI 单次扫描的结果，“抽取”是造成多次扫描结果累加的因素。

```
/* Self-cap and mutual-cap mode config */
const tsi_config_t hw_config =
{
    /* Self capacitance measurement config */
    configSelfCap.commonConfig.mainClock = kTSI_MainClockSlection_0,        // Set main clock
    configSelfCap.commonConfig.mode = kTSI_SensingModeSlection_Self,    // Choose SelfCap sensing mode
    configSelfCap.commonConfig.dvolt = kTSI_DvoltOption_0,                  // DVOLT (Vp-Vm)
    configSelfCap.commonConfig.cutoff = kTSI_SincCutoffDiv_0,               // Cutoff divider
    configSelfCap.commonConfig.order = kTSI_SincFilterOrder_2,              // SINC filter order
    configSelfCap.commonConfig.decimation = kTSI_SincDecimationValue_4,      // SINC decimation value
    configSelfCap.enableSensitivity = false,                                // SENS_BOOST = OFF
    configSelfCap.xdn = kTSI_SensitivityXdnOption_3,                        // Sens S_XDN
    configSelfCap.ctrim = kTSI_SensitivityCtrimOption_0,                    // Sens S_CTRIM
    configSelfCap.inputCurrent = kTSI_CurrentMultipleInputValue_0,          // Sens S_XIN = 1/8
    configSelfCap.chargeCurrent = kTSI_CurrentMultipleChargeValue_0,    // Sens S_XCH
```

$$NSTEP = \frac{Ci \times (Vp - Vm)}{vdd3v \times Cs \times S\_XIN \times S\_XCH}$$

SDK File: “**fsl_tsi_v5.h**”

```
typedef enum _tsi_current_multiple_input
{
    kTSI_CurrentMultipleInputValue_0 = 0U,  /*!<  S_XIN = 1/8 */
    kTSI_CurrentMultipleInputValue_1 = 1U,  /*!<  S_XIN = 1/4 */
} tsi_current_multiple_input_t;
```

图 60. 自电容模式下的 TSI 寄存器调整，灵敏度提升 = OFF

9.7.2 启用增强时的自电容灵敏度

通过“虚拟”去除一部分寄生电容，启用灵敏度提升功能可以提高灵敏度。因此，在启动灵敏度增强后，触摸在面板较厚的情况下仍然可以很好地工作。TSI 自电容模式通过取消外部固有电容来实现灵敏度的提高，并且其电容值可以从 2.5 pF 到 20 pF 的范围内取消，可在寄存器 TSI_MODE[S_CTRIM] 中配置。

例如，假设触摸电极的固有电容为 20 pF（可以通过 NSTEP 公式计算），则将 S_CTRIM 值设置为 5.0 pF 可以使有效的固有电容为 15 pF。由于触摸键的固有灵敏度是通过 ΔC/Cs 给出的，因此固有的电容越小，响应灵敏度就越高。启用此灵敏度增强功能后，灵敏度可以提高到 ΔCs/(Cs-S_CTRIM*(S_XDN/S_XCH))。

图 61 显示了启用了灵敏度增强功能的 TSI 自电容模式的方框图。灵敏度提升模块通过配置的内部电容器 C_{trim} 上的类似采样/电荷生成平均电流 I_{ctrim}。最终给 C_i 充电的电流为原始 I_{cx} 减去 I_{ctrim}。结果，外部触摸电极的电容似乎减去了 C_{trim}。顺便说一下，实际 C_{trim} 等于 $(X_{dn}/X_{ch})*C_{tri}$。

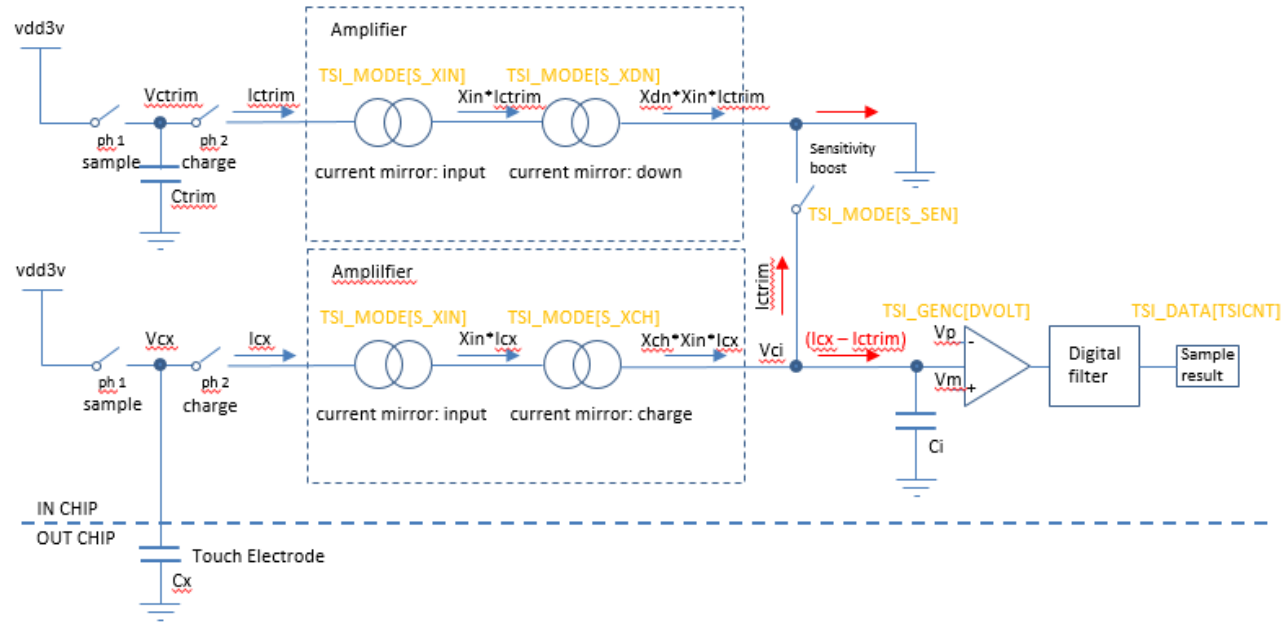


图 61. TSI 模块处于自电容模式, 灵敏度提升 = ON

```
/* Self-cap and mutual-cap mode config */
const tsi_config_t hw_config =
{
   /* Self capacitance measurement config */
   .configSelfCap.commonConfig.mainClock = kTSI_MainClockSlection_0,      // Set main clock
   .configSelfCap.commonConfig.mode = kTSI_SensingModeSlection_Self,    // Choose SelfCap sensing mode
   .configSelfCap.commonConfig.dvolt = kTSI_DvoltOption_0,                 // DVOLT (Vp-Vm)
   .configSelfCap.commonConfig.cutoff = kTSI_SincCutoffDiv_0,              // Cutoff divider
   .configSelfCap.commonConfig.order = kTSI_SincFilterOrder_2,             // SINC filter order
   .configSelfCap.commonConfig.decimation = kTSI_SincDecimationValue_4,     // SINC decimation value
   .configSelfCap.enableSensitivity = true,                                // SENS_BOOST = ON
   .configSelfCap.xdn = kTSI_SensitivityXdnOption_3,                       // S_XDN
   .configSelfCap.ctrim = kTSI_SensitivityCtrimOption_0,                   // S_CTRIM
   .configSelfCap.inputCurrent = kTSI_CurrentMultipleInputValue_0,         // S_XIN
   .configSelfCap.chargeCurrent = kTSI_CurrentMultipleChargeValue_0, // S_XCH
```

$$NSTEP = \frac{Ci \times (Vp - Vm)}{vdd3v \times (Cs - S\_CTRIM \times (S\_XDN \div S\_XCH)) \times S\_XIN \times S\_XCH}$$

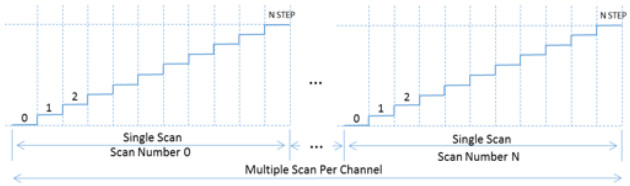
S_CTRIM: configurable, the capacitance to be removed.
S_XDN/S_XCH: configurable, the capacitance multiplier.
The actual capacitance to be removed is : $S\_CTRIM \times (S\_XDN \div S\_XCH)$

图 62. 在自盖式模式中启用灵敏度增强功能

9.7.3 TSI 扫描时间和结果累加

扫描时间确定转换结果的大小和时间。

TSI 支持每个通道进行多次扫描，这意味着 TSI 可以按顺序进行多次扫描以获得更好的 SNR 和分辨率。最终的扫描结果将在 TSI_DATA[TSICNT] 计数器中累积为 NSTEP 乘以扫描次数，并且扫描时间将为单个 TSI 扫描时间的倍数。请注意，较高的舍弃值会增加扫描次数，从而导致 TSI 计数器在物理上的累积更长，分辨率也更高。请注意，如果命令值大于 1，则 TSI 物理执行的扫描数小于硬件计算的扫描数，这可能有益于获得更高的分辨率。



TSI counter result accumulation

$$ScanNumber_{calculated} = \frac{Decimation^{Order}}{Cutoff} \qquad ScanNumber_{executed} = \frac{Decimation \times Order}{Cutoff}$$

图 63. TSI 扫描时间和累积

抽取，命令和截止的参数影响最终累计的扫描结果和总扫描时间。建议将命令值设置为 2，因为它可以节省扫描时间仍实现相同的数字扫描结果。

$$\text{ScanResult} = NSTEP \times ScanNumber_{calculated} = \frac{NSTEP \times Decimation^{Order}}{Cutoff}$$

$$\text{ScanTime} = Tnstep \times ScanNumber_{executed} = \frac{Tnstep \times Decimation \times Order}{Cutoff}$$

图 64. 扫描结果和扫描时间公式

```
/* Self-cap and mutual-cap mode config */
const tsi_config_t hw_config =
{
    /* Self capacitance measurement config */
    .configSelfCap.commonConfig.mode = kTSI_SensingModeSlection_Self,    // Choose SelfCap sensing mode
    .configSelfCap.commonConfig.cutoff = kTSI_SincCutoffDiv_0,           // Cutoff divider  = 1
    .configSelfCap.commonConfig.order = kTSI_SincFilterOrder_2,          // ORDER = 2
    .configSelfCap.commonConfig.decimation = kTSI_SincDecimationValue_8, // DECIMATION = 8
```

ScanNumberCalc (NSTEP multiple) = (8^2) / 1 = 64

ScanNumberExec (TSI hw really performed scans) = (8*2)/1 = 16

图 65. 抽取和截止设置

9.8 时钟生成和扩频时钟

TSI 时钟可以从可选择的异步内部时钟参考“主时钟”导出，可以进一步分频以获得最终的 TSI 扫描时钟频率。

基本和高级（SSC）时钟模式可用作时钟选项。

SSC（扩频调制时钟）可能有益于更高的 EMC 免疫力并减少 EMI。

- 基本：当 SSC_Mode = 10b 时，开关时钟直接从主时钟划分为基本时钟。
- 高级（SSC）：当 SSC_MODE = 00b/01b 时，然后从 SSC 模块生成开关时钟，作为高级时钟生成。

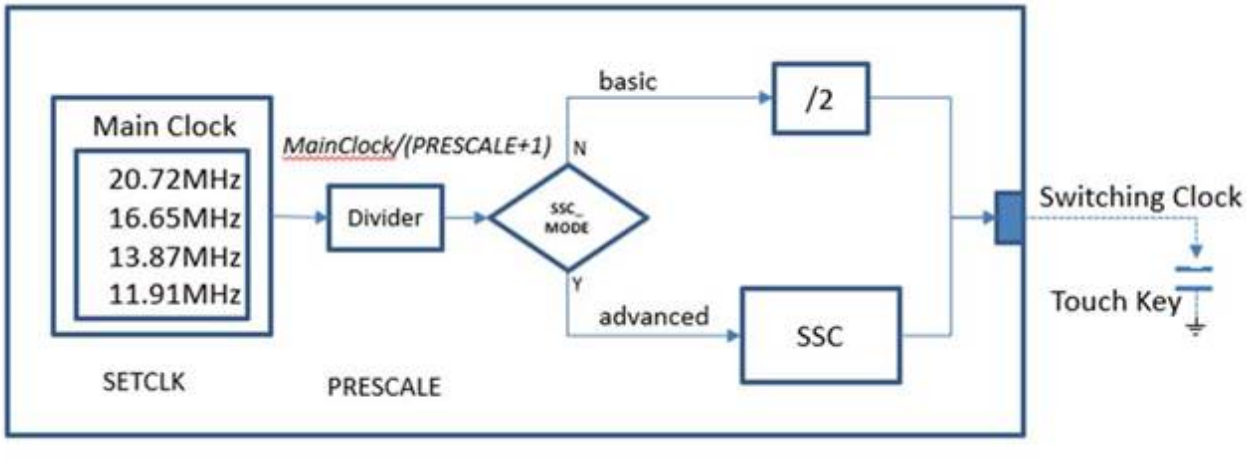


图 66. TSI v5 时钟生成框图

- 如果禁用了 SSC：

  TSI 开关时钟 = TSI_MainClock / (*SSC_PRESCALE_NUM+1)* / 2

- 如果启用了 SSC：

  TSI 开关时钟 = TSI_MainClock / (*SSC_PRESCALE_NUM+1)* / ((BASE_NOCHARGE_NUM+1)+(PRBS_OUTSEL+1)/ 2+ (CHARGE_NUM+ 1))

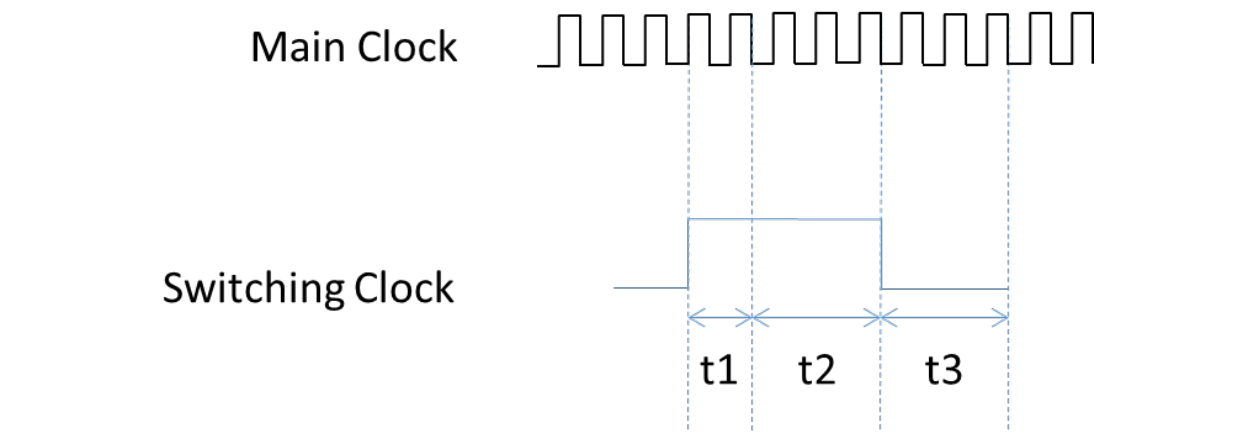


图 67. SSC 时钟原理

表 2. PRBS 随机时钟生成

Variable	Register	Descriptions
t1	TSI_SSC0[BASE_NOCHARGE_NUM]	SSCHighWidth
t2	TSI_SSC0[OUTSEL]	SSCHighRandomWidth
t3	TSI_SSC0[CHARGE_NUM]	SSLLowWidth

```
/* Self-cap and mutual-cap mode config */
const tsi_config_t hw_config =
{
   /* Self capacitance measurement config */
    configSelfCap.commonConfig.mainClock = kTSI_MainClockSlection_0,        // Set main clock
    configSelfCap.commonConfig.ssc_mode = kTSI_ssc_prbs_method,              // Select the SSC modulated clock output
    configSelfCap.commonConfig.mode = kTSI_SensingModeSlection_Self,    // Choose SelfCap sensing mode
    configSelfCap.commonConfig.chargeNum = kTSI_SscChargeNumValue_7,         // SSC T3 output bit0's period setting
    configSelfCap.commonConfig.noChargeNum = kTSI_SscNoChargeNumValue_5, // SSC T1 output bit1's period setting
    configSelfCap.commonConfig.prbsOutsel = kTSI_SscPrbsOutsel_2,             // SSC T2 output bit1's period setting
    configSelfCap.commonConfig.ssc_prescaler = kTSI_ssc_div_by_2,             // SSC clock prescaler
```

图 68. PRBS 寄存器 SSC 模式下的时钟设置

9.9 TSI IP 硬件寄存器调整 — 互电容模式

由于 TSI 功能的原理在互电容模式下不同，因此与自电容模式相比，修改的硬件模块与发送和接收电路一起使用，并且必须由用户配置的参数集也是不同的。

9.9.1 互电容模式下的灵敏度调整

默认寄存器配置是实验证明的，应该适合大多数应用程序。以“粗体”表示的参数是最重要的基本配置：

```
.configMutual.commonConfig.mainClock = kTSI_MainClockSlection_0, // Set main clock
.configMutual.commonConfig.mode = kTSI_SensingModeSlection_Mutual, // sensing mode = Mutual OK
.configMutual.commonConfig.dvolt = kTSI_DvoltOption_0, // Default: 0 (best) internal comparator
threshold voltage
.configMutual.commonConfig.cutoff = kTSI_SincCutoffDiv_0, // Divides the accumulated result, 0
recommended
.configMutual.commonConfig.order = kTSI_SincFilterOrder_2, // Length and multiply of the accumulated
result
.configMutual.commonConfig.decimation = kTSI_SincDecimationValue_4, // Multiple of real TSI scans
(longer acc.)
.configMutual.commonConfig.chargeNum = kTSI_SscChargeNumValue_4, // SSC clock settings
.configMutual.commonConfig.noChargeNum = kTSI_SscNoChargeNumValue_2, //SSC clock settings
.configMutual.preCurrent = kTSI_MutualPreCurrent_4uA, // Default: 4uA, controlling the Rx signal bias
voltage.
.configMutual.preResistor = kTSI_MutualPreResistor_4k, // Default: 4k, controlling the Rx bias
voltage; Urx > 0
.configMutual.senseResistor = kTSI_MutualSenseResistor_10k, // Rs resistor, used for translation of
the received U to I
.configMutual.boostCurrent = kTSI_MutualSenseBoostCurrent_0uA, // Sens boost factor minimized (No
benefits for SNR)
.configMutual.txDriveMode = kTSI_MutualTxDriveModeOption_0, // Default 0: (5V/-5V), 1: (0/5V) Tx
signal waveform gener
.configMutual.pmosLeftCurrent = kTSI_MutualPmosCurrentMirrorLeft_32, // Change this for sensitivity
tuning,.
.configMutual.pmosRightCurrent = kTSI_MutualPmosCurrentMirrorRight_1, // Default: 1
.configMutual.enableNmosMirror = true, // Default: true, Must be enabled
```

```
.configMutual.nmosCurrent = kTSI_MutualNmosCurrentMirror_1, // Default: 1, the same as
“MutualPmosCurrentMirrorRight
```

除时钟设置外，这还直接影响测量（开关时钟）的速度和累加结果。大多数参数应该保持默认值。

- 推荐 `kTSI_DvoltOption_0` 。
- 推荐 `kTSI_SincCutoffDiv_0` 。
- 推荐 `kTSI_SincFilterOrder_2`，我们可以尝试减小到 1，同时增加抽取率。
- `SincDecimationValue_4` 可以增加以获得更多的扫描次数，更长的累积时间和更高的分辨率。

无法选择控制发送通道信号的强度。

我们只能控制生成的 Tx 的信号的形状：

Tx 信号选项：

- ***kTSI_MutualTxDriveModeOption_0*** = 0U, /*!< TX drive mode is -5v~+5v, used in mutual-cap mode */
- ***kTSI_MutualTxDriveModeOption_1*** = 1U, /*!< TX drive mode is 0v~+5v, used in mutual-cap mode */

以下两个参数负责设置适当的 Rx 信号偏置（偏移）电压，在所有情况下均应为 V_{pre} > 0，以获取正常功能，请参见 图 69：

- `kTSI_MutualPreCurrent_4uA` （默认）
- `kTSI_MutualPreResistor_4k` （默认）

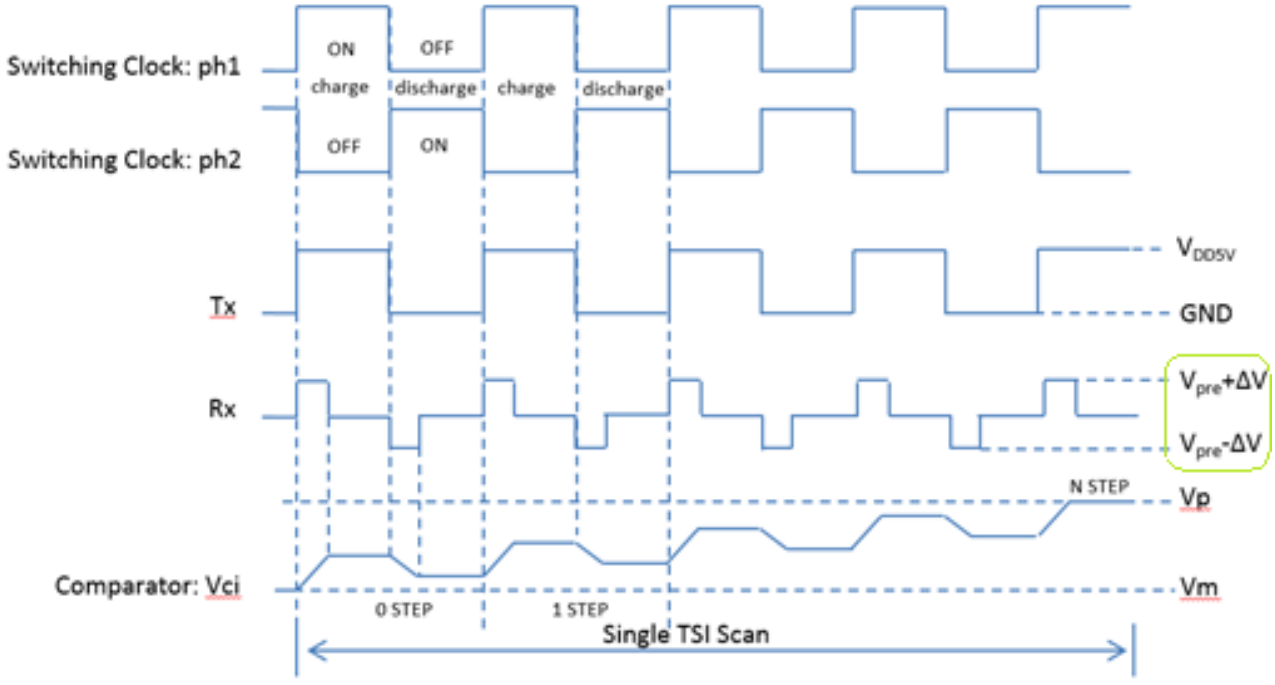


图 69. 接收偏移设置

我们可以测量 Rx 信号电平并正确调整 V_{pre}。或者我们可以尝试将发送通道切换为发送 0 - 5 V 电平，而不是 -5 V/5 V。

- `kTSI_MutualTxDriveModeOption_0` — 默认 0：（5 V/-5 V），我们可以尝试更改为 0 - 5 V。
- `kTSI_MutualSenseResistor_10k`（默认值）。该电阻 R_s 用于将接收到的 V_{rx} 电压转换为电流（I_{chg}/I_{dis}），以用于电流放大器输入。

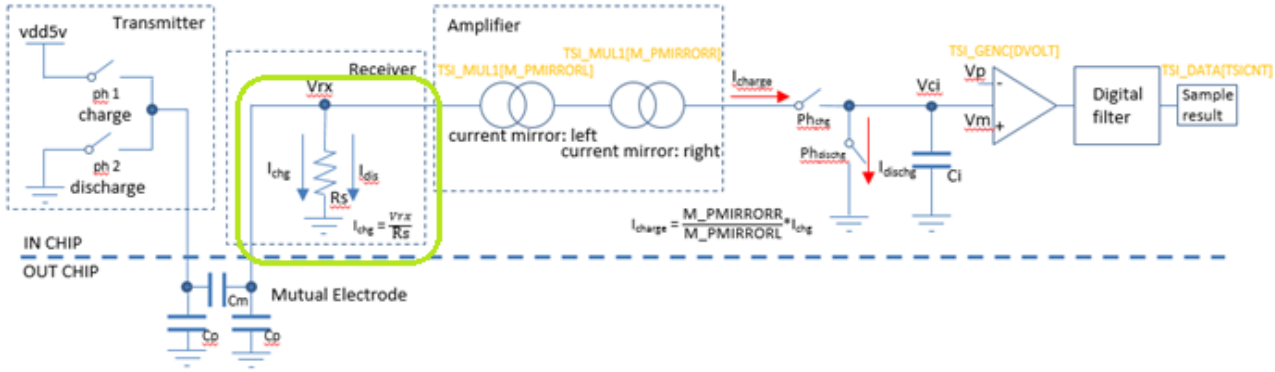


图 70. 互感电阻设置

由于 R_s 电阻的大小将接收到的电压信号转换为 I_{chg}，因此 R_s 的值也和灵敏度有关系。因此，使用这个值有意义，即使将 10 kΩ 作为默认值。更高的 R_s 值产生更高的触摸灵敏性。

- `kTSI_MutualSenseBoostCurrent_0uA` 是默认值，在互感式模式中与“灵敏度提升”有关，这与自感式模式不同。

使用默认设置（0uA）时，灵敏度增强功能非常弱。增加这个参数的数值会提高灵敏度，但对噪声的灵敏度也会提高，因此可能无法提高 SNR。我们可以尝试以较小步骤增加当前设置，以获得最佳结果。

- *kTSI_MutualPmosCurrentMirrorLeft_32* 是非常重要的参数，用于控制内部电流放大器的增益。数字越大，放大倍数越高。

因为放大系数由以下公式给出：

kTSI_MutualPmosCurrentMirrorLeft / kTSI_MutualPmosCurrentMirrorRight

- ***kTSI_MutualPmosCurrentMirrorRight_1*** — 建议使用默认值 = 1
- ***kTSI_MutualNmosCurrentMirror_1***

上面的两个设置应该保持相等，增加值可能会导致更快的响应。

- ***configMutual.enableNmosMirror*** = true

这个参数在所有情况下都必须为”true”。

9.9.2 互电容模式灵敏度调整说明

- 我们应该只在开始时使用粗体的参数。

```
.configMutual.pmosLeftCurrent = kTSI_MutualPmosCurrentMirrorLeft_32, // Change this for
sensitivity tuning,.
.configMutual.commonConfig.decimation = kTSI_SincDecimationValue_4, // Multiple of real TSI scans
(longer accumulation)
.configMutual.commonConfig.order = kTSI_SincFilterOrder_2, // Length and multiply of the
accumulated result, try “1” as well.
.configMutual.senseResistor = kTSI_MutualSenseResistor_10k, // Rs resistor, used for translation
of the received voltage to current
```

- 然后我们可以通过每一个小步骤来提高灵敏度，以达到最佳效果：

```
.configMutual.boostCurrent = kTSI_MutualSenseBoostCurrent_0uA
```

- 然后我们可以尝试通过以下方式调整 Rx 偏置电压：

```
.configMutual.preCurrent = kTSI_MutualPreCurrent_4uA, // Default: 4uA, controlling the Rx signal
bias voltage.
```

```
.configMutual.preResistor = kTSI_MutualPreResistor_4k, // Default: 4k, controlling the Rx bias
voltage; Urx must be > 0
```

- 保持 M_PMIRRORR 和 M_NMIRROR 相同。

```
kTSI_MutualPmosCurrentMirrorLeft = kTSI_MutualNmosCurrentMirror_1
kTSI_MutualPmosCurrentMirrorRight = kTSI_MutualPmosCurrentMirrorRight_1
```

- 请注意时钟设置会影响测量结果（累计长度）和计数器中的累计值。

10 屏蔽电极的设计原则

屏蔽方法用于消除温度位移，PCB 上的湿度或触摸控制面板上的水滴等环境影响。

- 问题：对于潮湿环境至关重要的是，新的触摸界面能够检测水滴与水层电容或手指电容之间的差异。
- 解决方法：键盘设计为带有“屏蔽”电极，可检测或补偿整体系统噪声或整体键盘电容。

10.1 软件屏蔽功能

NT 库提供软件屏蔽功能，此功能旨在检测由水滴引起的错误触摸并消除电容信号上调制的低频噪声。启用屏蔽功能后，将从相关的电极电容原始信号中减去屏蔽电容值。

该库在水滴和薄水膜下表现良好。在这些条件下，仅需要正确的校准即可准确检测触摸。

当检测到“无效触摸”时，屏蔽电极本身可以另外用作 GUARD 传感器。

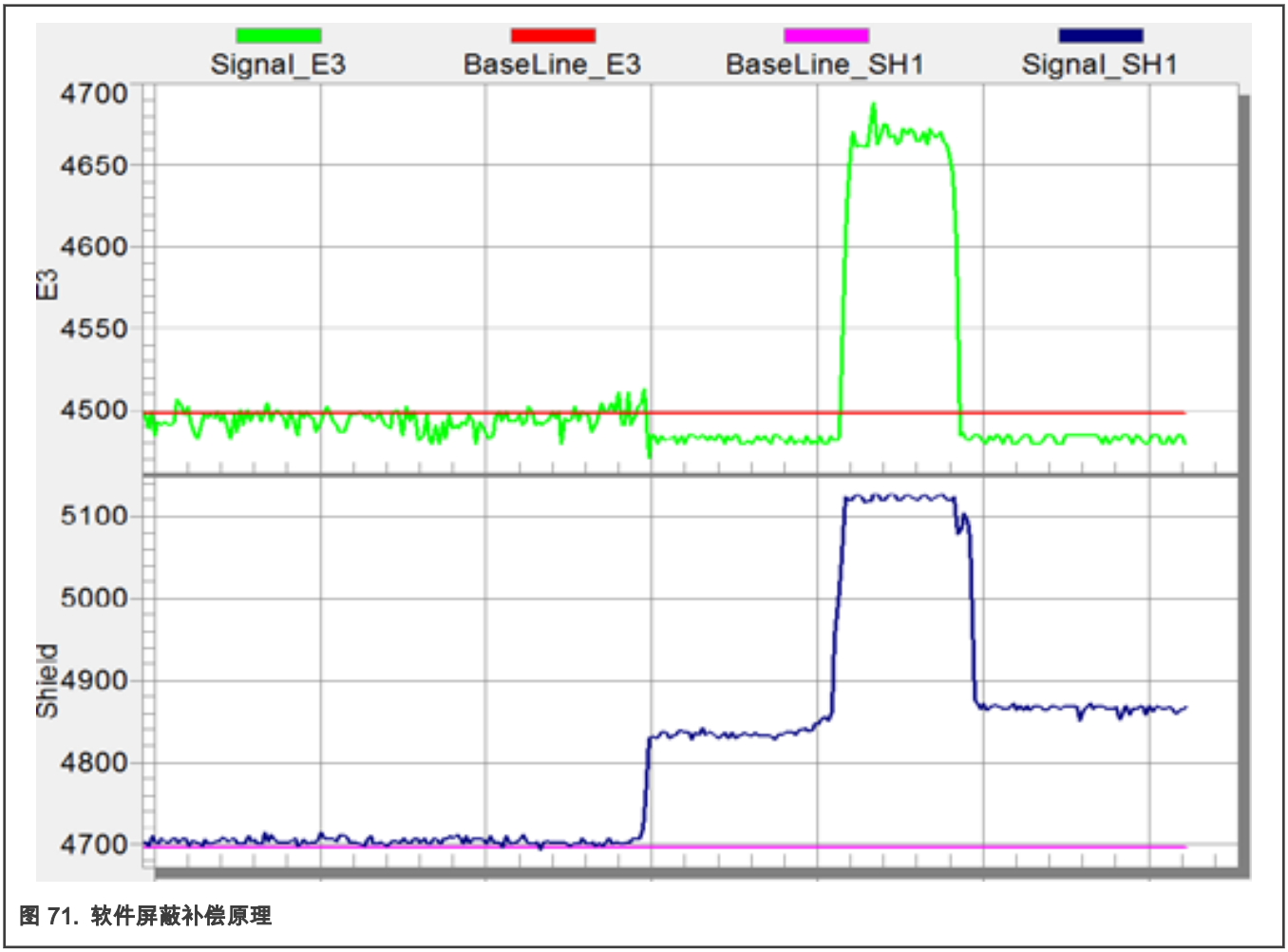


图 71. 软件屏蔽补偿原理

图 71 显示了电极即时信号及其屏蔽。从屏蔽信号（蓝色）可以看到，时间 = 10 秒，在板上放置了一层薄水膜。但是电极信号（绿色）停留在其基线（红色）附近。在时间 = 12.5 秒时，进行了手指触摸。由于从屏蔽层中的减法，所以电极的三角形看起来像是常规的触摸信号。

10.1.1 软件屏蔽设置

屏蔽电极主要是特殊电极或 PCB 图案，在正常条件下不接触，用于检测异常事件。

可以将屏蔽电极分配给常规电极。所有常规触摸电极都可以共享单个屏蔽电极或可以为不同的触摸感应电极分配不同的屏蔽电极。在特殊情况下，可以将常规电极用作其他电极的屏蔽，例如以补偿相邻按钮之间不必要的触摸信号串扰，以避免不必要的触摸检测。

通过“nt_electrode”结构定义中的参数来设置软件屏蔽设置，并且可以将其分配给每个电极。如果未定义“.shileding_electrode”或为 NULL，则表示未使用屏蔽功能，其余参数均被忽略。

```
const struct nt_electrode electrode_22 =
{
    .pin_input = EVB_BOARD_TOUCHPAD_0_ELECTRODE,
    .keydetector_interface = &nt_keydetector_usafa_interface,
    .keydetector_params.usafa = &keydec_usafa_keypad,
    .shielding_electrode = &electrode_27sh,
    .shield_threshold = 5,
    .shield_gain = 30,
    .shield_sens = 800,
};

/* Shield electrode */
const struct nt_electrode electrode_27sh =
{
    .pin_input = EVB_BOARD_SHIELD_ELECTRODE,
    .keydetector_interface = &nt_keydetector_usafa_interface,
    .keydetector_params.usafa = &keydec_usafa_keypad,
};
```

图 72. 软件屏蔽电极分配和配置

- “shielding_electrode”是用于屏蔽的电极，屏蔽电极具有自己的配置。如果在“SH”电极和“normal”电极上同时检测到常见信号变化，则可以激活软件补偿。
- “shield_threshold”是最小的共模信号阈值，其中屏蔽已激活。
- “shield_gain”是用于屏蔽电极信号的乘数（使屏蔽电极信号与“normal”触摸电极成比例）。
- “shield_sens”是用于常见信号偏移补偿的最大屏蔽电极偏移信号。意思是所有信号值 < shield_sens（800）将得到补偿（减去软件），并且如果值 > shield_sens （800）将不会得到补偿，并且可以在有效触摸下评估，在更糟的环境条件下。

在正常情况下，不希望触摸到屏蔽电极。我们可以设置一个按键检测器并为其设置“触摸”阈值，以使其既可以用作 GUARD 传感器，也可以用作传感器（水滴问题检测等）。然后阻止特定的键或完整的键盘。

10.2 软件屏蔽的优点和缺点

必须在软件设置中分别配置和调整每个电极的屏蔽强度。但是，如果有特殊需要或复杂的 PCB 布局，这可以提供更好的灵活性。

屏蔽电极占据标准的 TSI 通道，并按照常规触摸点击进行扫描。

10.3 硬件屏蔽功能（驱动屏蔽信号）

除了库中提供的软件屏蔽外，KE15z 设备还提供了另一种屏蔽方法。这是与先前描述的技术不同的技术，因为寄生电容补偿是在物理水平上完成的。KE15z 设备支持在 TSI ch12（PTC5）处提供一个硬件屏蔽信号输出。可以通过以下方式由 hw_config 中的单个 TSI 模块寄存器位启用：

```
.configSelfCap.enableShield = true,
```

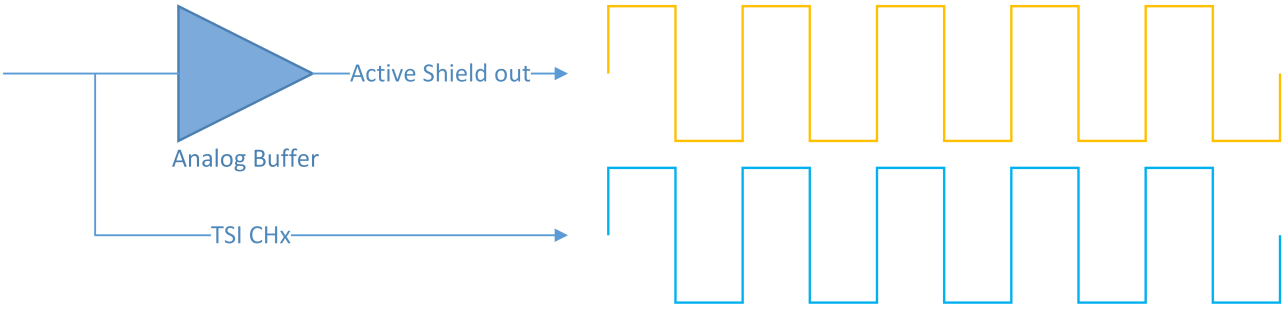


图 73. 主动屏蔽输出生成

硬件屏蔽信号是传感器充电信号的缓冲“副本”。（相同的幅度，频率和相位）

缓冲器提供足够的电流来驱动用作屏蔽电极的 PCB 上的网格敷铜的高寄生电容。

液滴和液流对电容传感器部分的影响，因为屏蔽电极是通过与传感器切换信号相同的电压驱动的，所以在触摸表面上由液滴添加的电容将被消除。

为了获得最佳的防水性能，要求驱动的屏蔽信号具有与传感器开关信号相同的“形状”（电压和相位）。

具有有源屏蔽功能的 PCB 必须经过精心设计，并且必须正确选择和调整分立的外部组件，以实现良好的功能。

主动屏蔽可以降低 PCB 的固有电容，从而提高标准触摸电极的总体灵敏度，这对于接近感应的实例可能是有益的。

11 结论

本文档介绍了在 FRDM-KE15z 板上演示的 NXP 触摸库和基于 FreeMASTER 的恩智浦 GUI 工具的基本用法和开发。详细介绍了 TSI 硬件电容式触摸感应原理和触摸灵敏度调整。最后一部分介绍了恩智浦触摸库和 TSI 硬件中可用的屏蔽方法。

12 参考文献

1. *NXP Touch Library Reference Manual*（文档 NT20RM）
2. *KE15 Touch Sensing Interface*（文档 KE15ZTSIUG）
3. *FRDM-TOUCH Quick Start Guide*（文档 FRDMTOUCHQSG）
4. *Kinetis KE1xZ Sub-Family Reference Manual*（文档 KE1XZP100M72SF0RM）
5. *Designing Touch Sensing Electrodes*（文档 AN3863）

13 修订记录

表 3. 修订记录

版本号	日期	说明
0	2020 年 1 月	初始版本

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2019 年 1 月 20 日

Document identifier: AN12709

arm